Design and Evaluation of a Context-Aware Orchestration Framework for Vulnerability Prioritization in Financial Infrastructure

Master Thesis

Alexandre Thorgal Meulien

Delft University of Technology

Design and Evaluation of a Context-Aware Orchestration Framework for Vulnerability Prioritization in Financial Infrastructure

by

Alexandre Thorgal Meulien

to obtain the degree of **Master of Science**

in **Computer & Embedded Systems Engineering**

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

at the Delft University of Technology,

to be defended publicly on Friday June 19, 2026 at 13:00.

Student number:	6297455	
Project duration:	November 2025 – June 2026	
Thesis committee:	Prof. dr. Georgios Smaragdakis,	TU Delft, chair
	Dr. Kaitai Liang,	TU Delft, daily supervisor
	Dr. Rui Wang,	TU Delft, daily co-supervisor
	Dr. Jérémie Decouchant,	TU Delft, committee member
	Jovan Aleksov,	ABN AMRO Bank N.V., company supervisor

Cover:	Abstract data-center lighting photograph by Krzysztof Kowalik via Unsplash, used under the Unsplash License.
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Preface

This thesis was written as part of the Master of Science programme in Computer & Embedded Systems Engineering (CESE) at Delft University of Technology. The research was conducted within the Cybersecurity Group at TU Delft in collaboration with ABN AMRO Bank N.V., where I worked within the Network Security Services (NSS) team. This project, which focused on the design and evaluation of a context-aware orchestration framework for vulnerability prioritization, gave me the opportunity to work at the intersection of software engineering, enterprise security, and AI-assisted decision-support systems.

I would like to express my sincere gratitude to my thesis committee chair, Professor Georgios Smaragdakis, for his guidance and oversight. I am particularly grateful to my supervisor, Kaitai Liang, and my co-supervisor, Rui Wang, for their constant support, technical feedback, and academic guidance. Their insights were essential for the development of both the architectural approach of the framework and the quality of this thesis.

I would also like to thank my company supervisor, Jovan Aleksov of ABN AMRO, for giving me the opportunity to conduct this research in a high-stakes operational environment and for his practical guidance throughout the project. I am grateful to the entire NSS team for their assistance, operational experience, and insights, which helped make this study possible. Navigating the balance between technical innovation and the strict reliability requirements of a financial institution has been a deeply rewarding challenge.

On a more personal note, I would like to thank my family and friends for their unwavering support, patience, and encouragement throughout my studies and during the writing of this thesis. I am especially grateful to my grandparents, whose financial support helped make my studies possible. Their generosity, belief in me, and continued encouragement have meant a great deal to me throughout this journey.

Finally, I would like to thank everyone who supported me, directly or indirectly, during my time at TU Delft. Completing this thesis marks the end of an important chapter in my academic life, and I am deeply grateful to those who helped me reach this point.

Alexandre Thorgal Meulien
Delft, June 2026

Abstract

Vulnerability prioritization in financial infrastructure depends on both public vulnerability signals and organization-specific context, such as affected assets, ownership, exposure, compliance status, and remediation constraints. Generic LLM-based systems can help analysts summarize and explain such information, but their direct use in regulated security workflows raises concerns about grounding, traceability, reliability, and control.

This thesis designs and evaluates a context-aware orchestration framework for LLM-supported vulnerability prioritization. The framework separates deterministic enterprise-data retrieval from LLM-based synthesis. For each analyst query, predefined workflows retrieve and join relevant vulnerability, asset, ownership, exposure, and compliance records, prune unnecessary metadata, and validate the generated output before presenting it as analyst-facing decision support.

The framework was evaluated against an autonomous-agent baseline in a controlled offline proof-of-concept environment using an approved static production-derived CSV snapshot from selected operational security systems. The benchmark covered 14 questions across three vulnerability-management use cases: contextual asset triage, risk justification, and remediation guidance. The parametric framework achieved a mean weighted accuracy of 98.4%, compared with 90.0% for the autonomous baseline, with the largest difference in contextual asset triage. It also consumed fewer tokens and showed less degradation under stochastic workflow noise.

The results suggest that constrained, context-aware orchestration can improve grounding, traceability, robustness, and operational efficiency for structured vulnerability-prioritization tasks. The findings should be interpreted as proof-of-concept evidence for the selected benchmark, not as evidence of production-scale deployment performance.

Keywords: vulnerability prioritization; large language models; context-aware orchestration; retrieval-augmented generation; financial cybersecurity; analyst decision support

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
API	Application Programming Interface
ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge
CESE	Computer and Embedded Systems Engineering
CISA	Cybersecurity and Infrastructure Security Agency
CMDB	Configuration Management Database
CSV	Comma-Separated Values
CTI	Cyber Threat Intelligence
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DORA	Digital Operational Resilience Act
EBA	European Banking Authority
EPSS	Exploit Prediction Scoring System
EU	European Union
FR	Functional Requirement
HITL	Human-in-the-Loop
ICT	Information and Communication Technology
IoC	Indicator of Compromise
JSON	JavaScript Object Notation
KEV	Known Exploited Vulnerabilities
KQL	Kusto Query Language
LLM	Large Language Model
NFR	Non-Functional Requirement
NIS2	Network and Information Security Directive 2
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NSF	Network Security Framework
PoC	Proof of Concept
RAG	Retrieval-Augmented Generation
ReAct	Reasoning and Acting
RQ	Research Question
SIEM	Security Information and Event Management
SLA	Service Level Agreement
SLO	Service Level Objective
SOC	Security Operations Center
SSVC	Stakeholder-Specific Vulnerability Categorization
VEX	Vulnerability Exploitability eXchange

Symbols

Symbol	Definition
W_{total}	Weighted accuracy score for a benchmark response
S_{server}	Score component for correct identification of affected servers
S_{owner}	Score component for correct identification of responsible asset owners
S_{field}	Score component for inclusion of expected supporting fields
$\mathcal{N}$	Stochastic noise profile used in the robustness evaluation
p_{hop}	Probability of injecting a missed tool-call hop
p_{drop}	Probability of dropping an expected server record
p_{param}	Probability of using an incorrect or malformed tool parameter
p_{hall}	Probability of injecting a hallucinated report line
p_{extra}	Probability of inserting an unnecessary tool call
p_{field}	Probability of omitting an expected output field
U_i	Uniformly distributed random variable used in noise injection
C_q	Estimated direct token cost per query
T_{in}	Number of input tokens
T_{out}	Number of output tokens
P_{in}	Price per one million input tokens
P_{out}	Price per one million output tokens
C_{annual}	Estimated annual direct token cost
Q_{day}	Number of daily analyst queries
H_{annual}	Estimated annual human-review cost
r_{review}	Fraction of outputs requiring manual review
C_{review}	Estimated cost of one manual review

1 Introduction

This chapter introduces the motivation, problem formulation, and objectives of this study. First, it explains why vulnerability prioritization in financial infrastructure requires an organization-specific context in addition to public vulnerability indicators. Second, it characterizes the problem addressed by the thesis, identifies the main research question and sub-questions, summarizes the contributions, and describes the scope and organization of the subsequent chapters.

1.1. Background and Motivation

Financial institutions must remediate software vulnerabilities in large and heterogeneous IT infrastructures. This thesis defines *vulnerability prioritization* as the selection of remediation actions to be performed first by integrating technical information about vulnerabilities with the organizational context. This task is challenging because security staff must respond rapidly to an increasing volume of disclosed vulnerabilities while managing limited capacity for remediation, continuity of service, and risk that vulnerable systems may support critical business functions [1], [2]. The prioritization problem is part of a broader European threat landscape in which ransomware, availability attacks, and threats to data continue to be major concerns for operators of critical digital services [3]. The relevance of this problem is especially pronounced for financial infrastructure, as ENISA's finance-sector threat landscape identifies financial organizations as a major target within the observed threat environment [4].

Traditional vulnerability management relies heavily on public-vulnerability signals. The Common Vulnerability Scoring System (CVSS) provides a standardized severity score, whereas the Exploit Prediction Scoring System (EPSS) estimates the likelihood that a vulnerability will be exploited in the field [5], [6]. Known exploitation sources, such as the CISA Known Exploited Vulnerabilities (KEV) catalog, provide further evidence that a particular vulnerability is being exploited by attackers [7]. These signals are valuable, but they do not fully answer the operational question faced by security teams: *which vulnerability should be remediated first in this specific organization?*

The missing component of *business context* includes information that influences the urgency of remediation, such as asset criticality, ownership, degree of exposure to the network, patterns of interdependencies, type of business function, availability of periods for maintenance, and use of compensating controls. The implementation of compensating controls reduces but does not eliminate the risk associated with a particular vulnerability. Consequently, the appropriate response to the same vulnerability on an isolated test server differs substantially from that required for the same vulnerability on an Internet-facing server that provides a critical banking service. Therefore, vulnerability prioritization is not only a technical scoring problem but also a contextual decision-support problem [8], [9].

Large Language Models (LLMs) create new opportunities for this type of decision support. LLMs can generate concise summaries of descriptions of vulnerabilities, explain the results of technical investigations, and convey information to security analysts and remediation personnel in a more readily understandable format. However, generic LLM-based systems require substantial adaptation to ensure their correct use in enterprise security workflows. They require *grounding*, which in this thesis means that the generated outputs should be based on retrieved and inspectable evidence rather than on unsupported model assumptions. Without grounding, an LLM may generate seemingly plausible but factually incorrect assertions concerning exploitability, exposure, ownership, or compliance status [10], [11].

This limitation is particularly important in regulated financial settings. Decision support for analysts must remain transparent, constrained, and consistent with organizational policies. Requirements for regulation and governance also limit which data may be processed, where it may be processed, and to what degree an AI-based system may operate autonomously in security processes [2], [12]. Consequently, an LLM cannot simply be connected to enterprise data sources and permitted to operate without constraints. Instead, the architecture of the system must control the types of information retrieved, the manner in which that information is organized, and the conditions under which probabilistic inference is permitted.

1.2. Problem Statement

The direct application of generic LLM-based systems to enterprise vulnerability prioritization is constrained by three engineering challenges.

First, generic LLMs lack access to the internal metadata required to convert public information about vulnerabilities into organization-specific risk interpretations. This results in a *semantic gap* in which signals describing the technical properties of a vulnerability are insufficient to support decisions requiring knowledge of affected assets, ownership, exposure, compliance status, and business function.

Second, enterprise security data is generally too large and heterogeneous to permit direct insertion into a prompt. Many fields of raw records for configuration management, vulnerability scans, compliance assessments and asset ownership are irrelevant to a specific analyst task. Excessive inclusion of context increases costs, latency, and consumption of tokens [13] and does not necessarily improve the use of relevant information.

Finally, the application of probabilistic AI techniques to prioritize vulnerabilities in a financial environment requires mechanisms for traceability and control. Retrieval-augmented generation can ground model outputs in external evidence [14], but retrieval pipelines can fail during retrieval, augmentation, or generation [15]. Tool-use approaches, such as Toolformer and ReAct, demonstrate that LLMs can interface with external components but must be appropriately constrained for application to enterprise security workflows [16], [17]. These limitations suggest the need for controlled methods of data retrieval, integration, and validation, rather than unconstrained interaction with an AI model.

This thesis addresses these challenges by designing and evaluating a *context-aware orchestration framework*. In this study, context-aware orchestration refers to the controlled coordination of data retrieval, metadata pruning, LLM-based reasoning, and output validation for vulnerability prioritization. The goal is not to automate patching or remediation execution but to provide analysts with well-supported, traceable, and operationally useful assistance for prioritizing vulnerabilities.

1.3. Thesis Objective and Research Questions

The objective of this thesis is to design, implement, and evaluate a context-aware orchestration framework for vulnerability prioritization in financial infrastructure. The framework integrates information from heterogeneous sources to generate compact, task-specific representations of the enterprise context and applies LLM-based reasoning only after appropriate evidence has been identified and integrated.

The result is a security analyst decision-support system that helps answer questions such as which assets are affected by a vulnerability, why a particular vulnerability should receive priority in a given environment, and what information should be conveyed to asset owners to support remediation actions. The performance of this framework is assessed for three representative activities of vulnerability management: contextual asset triage, risk justification, and remediation guidance.

What differentiates this thesis from prior approaches is not the development of an additional vulnerability score or standalone LLM prompt but the design and evaluation of a governance-aware orchestration architecture for vulnerability prioritization. Previous studies have addressed vulnerability scoring, retrieval augmentation, LLM tool integration, and AI governance as distinct concerns. In contrast, this framework integrates these concerns into a single decision-support pipeline. Deterministic retrieval and processing of enterprise data, followed by bounded LLM-based synthesis, support an architecture that aligns with the constraints of regulated financial-security workflows.

The main research question of this thesis is as follows:

> **MRQ:** How can LLM-supported vulnerability prioritization be made grounded, efficient, and operationally usable in a regulated financial environment?

This main question is addressed through the following sub-questions:

RQ1: How can heterogeneous security and organizational data be selectively retrieved and transformed into context usable for vulnerability prioritization?

RQ2: How well does the proposed orchestration framework perform on representative vulnerability-management tasks in terms of accuracy, operational overhead, estimated correction effort, and operational cost?

RQ3: Which architectural safeguards are required to make LLM-supported vulnerability prioritization usable in a regulated financial environment?

RQ1 addresses the challenges of data and context construction. RQ2 assesses the effectiveness of the proposed framework in terms of quantifiable metrics of system performance and cost. In this study, operational overhead is measured using proxy metrics, including tool invocations, token consumption, response latency, and projected review or correction costs. This evaluation does not include a user study and therefore provides no direct measure of analyst workload, cognitive effort, level of trust, or degree of acceptance. RQ3 identifies architectural requirements for application in a regulated financial environment, including requirements for traceability, control of data boundaries, validation, and levels of human supervision.

1.4. Contributions

This thesis makes four main contributions.

1. A context-aware orchestration framework that combines deterministic enterprise-data retrieval with bounded LLM-based reasoning for vulnerability prioritization.
2. A selective metadata extraction and normalization approach for integrating heterogeneous information from vulnerability records, asset metadata, ownership data, and compliance context.
3. An implementation and evaluation of the framework across three representative vulnerability-management scenarios: contextual asset triage, risk justification, and remediation guidance.
4. A discussion of the operational limits of LLM-supported vulnerability prioritization and the role of analyst oversight in regulated financial environments.

These combined efforts position this thesis as an applied systems engineering investigation of the governance-aware orchestration of large language models for vulnerability prioritization. The central contribution is the integration of retrieval control, contextual pruning, output validation, and analyst-oriented synthesis, rather than reliance on any individual scoring rule, prompt template, or isolated retrieval method.

1.5. Scope and Thesis Outline

The scope of this thesis is restricted to analyst-facing decision support. The framework does not automatically apply patches, modify configurations, invoke incident-response procedures, or execute remediation workflows. Instead, it supports vulnerability prioritization by retrieving relevant context, structuring evidence, and generating analyst-facing explanations.

The implementation was evaluated in a bounded proof-of-concept setting using selected server-side vulnerability-management scenarios and an approved static production-derived CSV snapshot. The snapshot contained selected vulnerability, asset, and compliance records obtained from operational systems, with identifying fields partially obscured for confidentiality while preserving the relationships required for evaluation. The system was not connected to live production databases, scanners, CMDB systems, compliance platforms, or production servers during benchmark execution. Therefore, the findings should be interpreted as evidence from a controlled offline benchmark rather than as evidence of production-scale deployment performance.

The remainder of this thesis is structured as follows. Chapter 2 discusses background and related work. Chapter 3 defines the requirements and evaluation approach. Chapter 4 presents the architecture of the proposed framework. Chapter 5 describes the proof-of-concept implementation. Chapter 6 evaluates the framework across the selected use cases. Chapter 7 discusses the results, limitations, and trade-offs. Finally, Chapter 8 concludes the thesis and outlines directions for future work.

2 Background and Related Work

This chapter provides the technical and regulatory context necessary to position the proposed framework. It begins with a discussion of conventional vulnerability management and the shortcomings of severity-based prioritization. It then describes context-aware prioritization, the application and limitations of large language models in cybersecurity, retrieval-augmented generation and orchestration, and the requirements for financial system governance. Finally, it identifies the research gap addressed by this study.

2.1. Traditional Vulnerability Management

Vulnerability management is the process of identifying, assessing, prioritizing, and remediating weaknesses in software and infrastructure. In large enterprise environments, this process is not a one-time technical activity but rather a continuous operational practice. Security teams must respond to newly disclosed vulnerabilities across heterogeneous assets while working within limited remediation capacity, service availability constraints, and operational risk requirements. This challenge is particularly significant in the regulated financial environment. In these settings, vulnerable systems may support critical business operations and must be managed in accordance with the requirements for resilience, operational control, and cybersecurity compliance [1], [18].

The Common Vulnerabilities and Exposures (CVE) system provides standardized identifiers for publicly known vulnerabilities [19]. These identifiers allow vulnerabilities to be referenced consistently across scanners, advisories, vulnerability databases and remediation workflows. However, CVE identifiers do not by themselves express the severity, exploitability, or organizational impact of a vulnerability. For this reason, CVE information is commonly combined with additional scoring and prioritization mechanisms.

Additional methods for scoring and prioritizing vulnerabilities are routinely applied in practice. The use of the Common Vulnerability Scoring System is one of the most widely used approaches for communicating the severity of vulnerabilities [5], [20]. CVSS version 4.0 includes several groups of metrics that describe the technical characteristics of vulnerabilities and permit comparisons between vulnerabilities. This system is well-suited for the initial classification and reporting of vulnerabilities. However, it does not represent a complete assessment of the risk to an organization.

The explicit treatment of the Common Vulnerability Scoring System as a measure of severity, rather than of overall risk, further supports the need for integration with information specific to an organization [21]. Even when environmental metrics are available, the practical use of CVSS still depends on an organization-specific interpretation. A severity score alone does not automatically determine whether a vulnerability is urgent in a particular enterprise environment.

Exploitation likelihood metrics extend severity scoring to provide estimates of the likelihood that a vulnerability will be exploited in the wild within a short timeframe [6]. The EPSS model was introduced as a data-driven approach for estimating the probability that a vulnerability will be exploited in the wild, thereby complementing severity-based scoring with exploitation likelihood evidence [22]. This distinction is important because technical severity and likelihood of exploitation are not equivalent. Some highly severe vulnerabilities have limited evidence of exploitation, whereas other vulnerabilities with less severe technical ratings require immediate attention because exploitation is probable or has already occurred. The operational response to vulnerabilities is strongly influenced by evidence of active exploitation [7].

The catalog of known exploited vulnerabilities maintained by the CISA provides a practical basis for prioritizing remediation when exploitation is observed [7]. Similarly, the NIST guidance on enterprise patch management frames patching as a broader process of identifying, prioritizing, acquiring, installing, and validating patches for organizational assets [1]. The NIST guidance on continuous information security monitoring complements this view by treating vulnerability and configuration information as part of ongoing organizational risk awareness [23]. Structures such as MITRE ATT&CK further relate vulnerability information to adversary behavior and defensive actions [24], [25].

Despite their value, public scoring and exploitation signals are insufficient when used alone. A high severity vulnerability will result in a limited immediate impact if the affected system is isolated or not critical. In contrast, a low-severity vulnerability would require urgent remediation if the affected asset is internet-facing, represents a critical business function, or lacks compensating controls. Consequently, the effective determination of priorities requires the integration of public information about vulnerabilities with details concerning internal conditions.

This limitation motivates the transition from vulnerability scoring to context-aware decision support. Public vulnerability identifiers, severity scores, exploitation-likelihood estimates, and known-exploitation catalogues provide a common basis for describing vulnerabilities, but they do not determine remediation urgency in a specific organization. For this thesis, their main limitation is that they require enterprise context before they can support prioritization decisions.

2.2. Context-Aware Vulnerability Prioritization

Context-aware vulnerability prioritization extends traditional vulnerability scoring to incorporate the environment in which a vulnerability occurs [8], [9]. A given vulnerability has different operational implications for different organizations and assets. The urgency of remediation depends on factors such as external exposure of affected systems, support for critical business processes, presence of compensating controls, and feasibility of remediation without disruption of essential services. Therefore, prioritization requires the integration of public information about vulnerabilities with details about the organizational context.

This important distinction is also reflected in risk-based vulnerability management approaches, where prioritization depends not only on technical severity but also on exploitability, exposure, mission impact, and stakeholder-specific response decisions [8], [9], [20], [21]. Public methods for describing the characteristics and effects of vulnerabilities provide standardized information for comparing the relative severity of different vulnerabilities [5], [6], [7]. Indicators of potential or actual exploitation provide additional information about the likelihood and impact of vulnerability exploitation. However, this data does not directly represent the role of the affected asset within a specific enterprise.

Remedial actions required for a noncritical internal test system would differ substantially from those required for the same vulnerability on an Internet-facing production system supporting a critical banking operation. Consequently, context-aware prioritization depends on internal information that explains how a vulnerability affects the organization. This includes who owns the affected asset, whether the asset supports a critical service, how exposed it is, whether compensating controls exist, and whether remediation can be performed without disrupting essential operations. In this study, these factors are collectively referred to as the business context. This information enables analysts to convert technical descriptions of vulnerabilities into decisions regarding appropriate remedial actions.

Frameworks directed toward remediation decisions reflect the shift from methods of expressing vulnerability severity to methods of expressing the requirements for remediation [8], [9]. For example, factors used to determine the appropriateness of different remedial actions include the status of exploitation, magnitude of technical impact, degree of relevance to mission objectives, and level of exposure to the external environment. Consequently, these methods are appropriate for application in enterprise settings, where it is recognized that the requirements for remediation depend on the objectives of stakeholders and conditions of the operational context, rather than solely on measures of the severity of a particular vulnerability.

Recent studies have also illustrated the trend toward context-aware prioritization. Studies on vulnerability prioritization highlight major classes of prioritization metrics that include contextual factors, predictive indicators, and methods of aggregation [26]. The use of context-aware models, such as the CAVP, further illustrates the potential for improvement achieved by incorporating additional contextual and temporal information into the score generation process [27].

However, context-aware prioritization introduces an important engineering challenge. The relevant context is generally distributed among heterogeneous systems of enterprise information, including configuration management databases, vulnerability scanners, monitoring systems, ownership records, and inventories of IT services. This information is often incomplete, inconsistent, or too voluminous to permit direct use in an analyst-oriented workflow. As a result, the implementation of context-aware prioritization requires the additional acquisition of information and the development of methods to select, normalize, and convey appropriate aspects of the context in a manner that supports analyst-backed decision-making. This represents the core of the proposed framework.

Existing methods of prioritization reflect the importance of context but do not provide an architecture for the systematic retrieval and application of context in a controlled, auditable, and scalable manner. The framework explored in this study addresses this limitation by incorporating a level of system orchestration that selectively integrates enterprise metadata with LLM-based reasoning to support decision-making by analysts. The resulting distinction between public indicators of vulnerability and internal aspects of the enterprise context is illustrated in Figure 2.1.

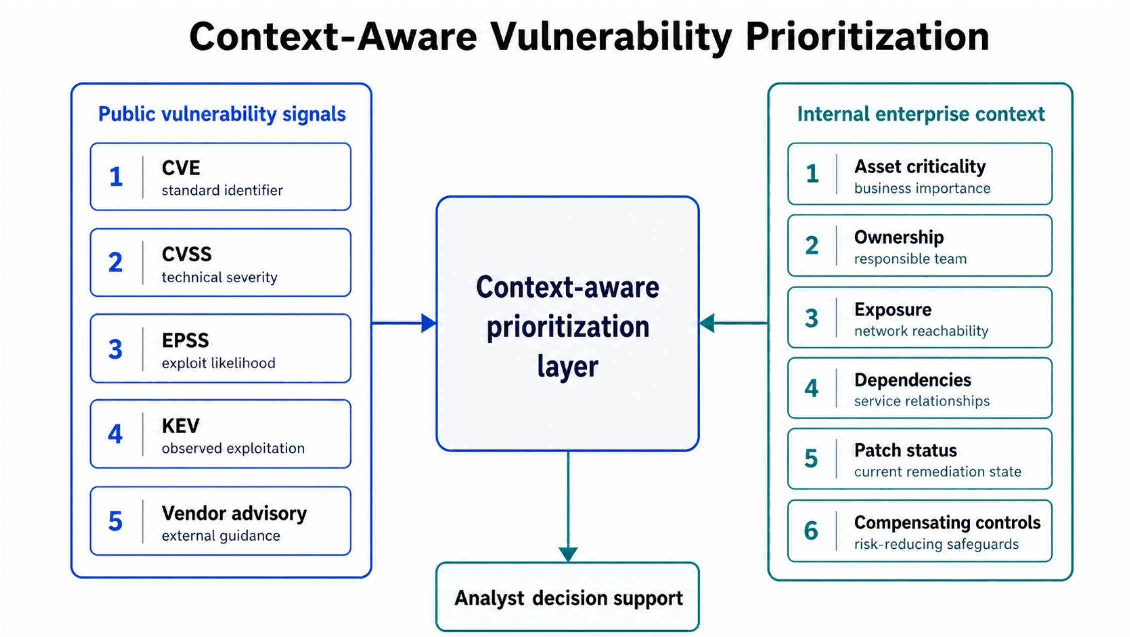

Figure 2.1: Context-aware vulnerability prioritization combines public vulnerability signals with internal enterprise context. Public signals describe severity, exploit likelihood, and observed exploitation, while internal context determines how those signals are interpreted for assets in a specific organization.

For this thesis, context-aware prioritization is considered primarily a problem of data integration and decision support rather than solely a technical exercise of severity scoring.

2.3. Large Language Models in Cybersecurity

Large language models are becoming increasingly important for cybersecurity because many security tasks require the interpretation of heterogeneous textual and technical information. Security analysts routinely work with descriptions of vulnerabilities, advisories, log fragments, reports of threat intelligence, metadata describing assets and methods for remediation, and other artifacts requiring summarization, comparison, translation to operational terms, and integration with overall risk assessments. Recent studies have demonstrated that large language models are being applied to many tasks in cybersecurity, including the analysis of threats, vulnerabilities, malware, secure software development, and operations support [28], [29], [30]. Therefore, LLMs are well-suited to support analysts with certain tasks because they can process natural language, generate explanations, and integrate information from multiple sources.

One important application area involves extracting meaning from reports, indicators, and descriptions of adversaries and vulnerabilities and applying the resulting information to operational decisions. Benchmarks such as CTIBench evaluate LLM performance on cybersecurity-specific reasoning and information extraction tasks [10]. Broader cybersecurity benchmarks, such as SECURE and CyberSecEval, further confirm that LLM performance should be evaluated on domain-specific security tasks rather than inferred from general language model capability alone [31], [32]. This study is applicable to vulnerability prioritization and illustrates that general-purpose LLMs can support security analysis only when their performance is evaluated in realistic settings. Taken together, these benchmarks converge on the same point: LLMs may be useful for cybersecurity analysis, but their reliability must be established through task-specific evaluations rather than assumed from general-purpose performance.

In addition, LLMs can support workflows for managing vulnerabilities by generating summaries, explanations, and texts appropriate for remediation activities. For example, they can be used to explain the importance of a vulnerability, summarize affected systems, or prepare communications for remediation teams. The use of methods such as LLM4Vuln [33] is appropriate because it isolates aspects of vulnerability reasoning from other factors such as retrieval of knowledge, design of prompts, and provision of contextual information. These capabilities are valuable because vulnerability prioritization represents both a problem of ranking and communication.

Analysts must justify the importance of certain vulnerabilities and convert technical results into operational decisions. LLMs can reduce the effort required by analysts to draft such explanations. The common capabilities and limitations of LLMs for supporting cybersecurity decision-making are summarized in Table 2.1.

Table 2.1: LLM capabilities and limitations in cybersecurity decision support.

Capability	Example in vulnerability management	Limitation
Summarization	Condensing vulnerability descriptions, advisories, and affected-system information	May omit relevant context or overgeneralize findings
Explanation	Producing risk justifications for analysts or remediation teams	May generate unsupported or overconfident reasoning
Information synthesis	Combining vulnerability details with retrieved asset and ownership context	Requires accurate, current, and relevant context retrieval
Communication support	Drafting remediation summaries or analyst-facing notes	Requires human review before operational use

Although they are capable of great versatility, large language models introduce challenges to reliability, which are especially important for cybersecurity. Their outputs may include hallucinated information, unsupported statements, or apparently valid but incorrect reasoning. Hallucination represents a general limitation of large language models but is of special concern for security applications in which erroneous information can lead to inappropriate efforts at remediation or unjustified priorities for system protection [11]. As a result, an unsupported claim concerning exploitability, degree of asset exposure, or adequacy of compensating controls directly affects the risk interpretations by analysts.

Other limitations of large language models are also a major concern. In general, these models lack awareness of the current conditions of internal infrastructure, asset ownership, degree of exposure to the network, and status of system patches. The absence of access to these conditions limits the ability of the models to reason from information on publicly available vulnerabilities and from text included in the input prompts. This closely reflects the gap in meaning previously discussed in Section 2.2: priorities for vulnerability management require knowledge of the internal conditions of the enterprise but are not inherently related to these conditions by large language models.

Additional studies have also demonstrated that the application of large language models to cybersecurity activities can result in new modes of failure. Experience with large-language-model-based systems for cyber threat intelligence illustrates that such systems are susceptible to errors of incorrect information extraction, misleading or inconsistent interpretations of results, and excessive confidence in the validity of conclusions in the absence of adequate controls on or validation of model outputs [10], [34]. Subsequent studies have shown that LLM performance varies across cybersecurity tasks and that operational use requires task-specific validation and characterization [31], [32], [33]. These requirements are of particular importance for regulated financial institutions, where it is necessary to provide complete documentation of the methods used and maintain full control over all system activities.

Therefore, large language models should not be regarded as independent systems for making decisions regarding priorities for vulnerability management. Instead, they provide the principal benefits of synthesis, explanation, and interaction. The selection of factual information and retrieval of data must be achieved by the controlled components of the system. This leads to an organization of system operations in which the processing of information by large language models is separated from the deterministic retrieval of information on the conditions of the enterprise. This mode of operation permits the full utilization of the capabilities of large language models for the purposes of explanation while minimizing the risks of the generation of unsupported or unverifiable results.

In this thesis, LLMs are treated as components of analyst-facing decision support rather than as autonomous security operators. The framework presented in the subsequent chapters employs LLM-based reasoning to enable triage, justification of risk, and recommendations for remediation, but only after the appropriate context has been obtained and integrated by the orchestration layer. This reflects the view that LLMs are effective for cybersecurity applications only when they are well-constrained, well-scoped, and well-grounded, and are not sufficient for high-risk prioritization as unconstrained tools. This is consistent with the well-documented limitations of existing systems for cybersecurity analysis.

2.4. Retrieval-Augmented Generation and Architectural Orchestration

Retrieval-augmented generation (RAG) and architectural orchestration provide controlled access to external information for use with LLM-based reasoning, thereby overcoming the fundamental limitations of generic LLM systems [14], [15]. This approach supports the application of retrieval and orchestration techniques to the context of vulnerability prioritization.

Useful prioritization output cannot be generated from model parameters or public vulnerability descriptions alone. It also requires organization-specific information about assets, ownership, exposure, remediation constraints, and compliance status. Metadata describing assets, records of vulnerabilities, information on ownership, indicators of exposure, and constraints on remediation represent the types of information that must be obtained from outside the model. For this reason, access to external information is achieved with full control and in a well-structured manner using retrieval-augmented generation and architectural orchestration techniques.

2.4.1. Retrieval-Augmented Generation

Retrieval-augmented generation involves providing a large language model with externally retrieved information before producing an answer to a question. Rather than relying solely on internal parameters, the system first retrieves relevant documents, records, or structured data and incorporates the retrieved information into the generated response [14]. This approach is appropriate for domains in which answers depend on current or domain-specific information that is not available in the training data for the language model. In particular, this is essential for cybersecurity applications, in which knowledge continuously evolves. Descriptions of vulnerabilities, information on exploitability, status of assets, and priorities for remediation change over time, and a static model cannot represent the current status of an enterprise environment. Thus, retrieval-augmented generation represents a means of anchoring outputs to external sources of information and reducing the model's reliance on unsupported knowledge.

However, retrieval-augmented generation is not a complete solution to this problem. The quality of retrieval determines the extent to which the model receives relevant and sufficient information to produce accurate results. Poor retrieval can lead to incomplete or incorrect answers. The evaluation of systems that apply retrieval-augmented generation demonstrates that failures of any type can occur at any stage of the pipeline [15]. Therefore, retrieval, augmentation, and generation must be evaluated in an integrated manner. Additional experience with methods for quantifying performance for retrieval-augmented generation confirms that overall system quality depends on multiple factors, including the relevance of retrieved information, quality of retrieved context, degree of faithfulness to the original information, and relevance of final results [35], [36].

For applications involving the prioritization of vulnerabilities, it follows that the simple addition of a vector database or document retrieval system will be insufficient. The system must also select appropriate sources for retrieval, determine which records to retrieve, and define how the retrieved information should be integrated to support subsequent reasoning.

2.4.2. Context Management and Pruning

Context management is a central challenge in applying RAG to enterprise security data. Large organizations maintain extensive security and asset management datasets; however, only a small subset of that information is relevant to a given vulnerability question. Including excessive context in prompts increases token consumption, latency, and cost. More importantly, long prompts do not guarantee improved reasoning. Studies on long-context language models have shown that models may fail to use relevant information effectively when it is placed within a large context window, a phenomenon often described as being “lost in the middle” [13].

This limitation is directly relevant to the prioritization of vulnerabilities. A naive implementation could retrieve entire database rows, large lists of affected assets, or broad vulnerability records and place them in a model prompt. Such an approach may include correct information, but it also introduces irrelevant or redundant data that can make the model output less reliable. Therefore, context-aware systems require pruning mechanisms that select only the information required for the current task. Related work on adaptive retrieval, such as Self-RAG, also motivates the idea that retrieval should be selective rather than applied indiscriminately [37].

In this study, context pruning refers to the process of reducing retrieved enterprise data to the subset of metadata required for a specific prioritization decision. For example, a triage query may require the affected asset identifier, ownership information, exposure status, and vulnerability severity but not the full historical change record of the asset. Similarly, a remediation guidance query may require patch status, deadline, and affected service information but not unrelated configuration attributes. By narrowing the input to relevant fields, the system can reduce token usage while improving reasoning context clarity.

Figure 2.2 summarizes the role of retrieval, pruning, reasoning, and validation in the proposed orchestration pattern.

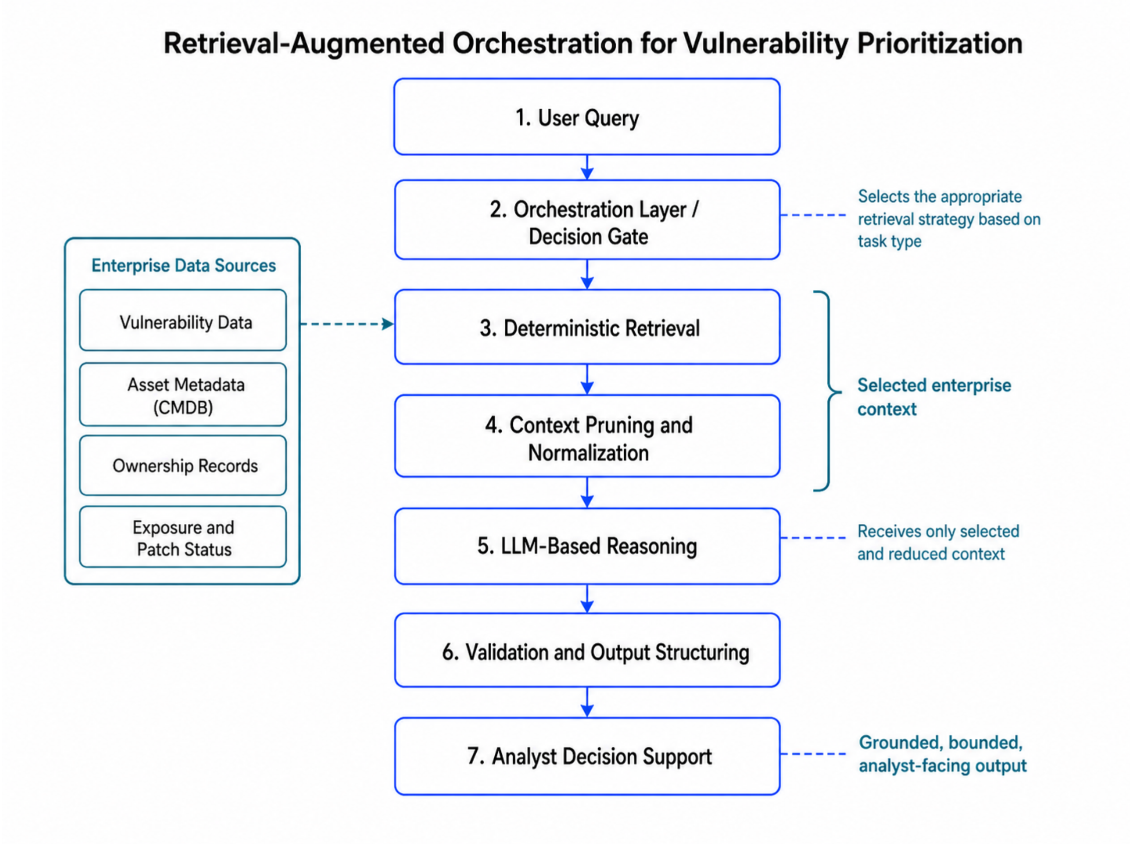

Figure 2.2: Retrieval-augmented orchestration pattern for vulnerability prioritization. A user query is first interpreted by an orchestration layer, which determines the required data sources and retrieval strategy. The relevant enterprise context is then retrieved, pruned, and normalized before being passed to the LLM-based synthesis component. The generated output is validated and structured as analyst-facing decision support rather than autonomous remediation.

2.4.3. Deterministic and Probabilistic Orchestration

Architectural orchestration extends RAG to coordinate multiple system components rather than treating a large language model as an isolated interface. In an orchestrated system, the model may be responsible for interpreting user intent or generating explanations, whereas deterministic components perform data retrieval, schema validation, policy enforcement, and output formatting. This division of responsibilities is particularly important for enterprise security, in which not all tasks should be subjected to probabilistic reasoning. Toolformer and ReAct illustrate how LLMs can be connected to external tools and intermediate reasoning steps, respectively [16], [17]. These approaches are valuable because they demonstrate the feasibility of combining model outputs with external actions rather than being limited to a single-turn text generation. Subsequent evaluations of performance on tasks of tool use and agent-based reasoning [38], [39], [40], [41], [42] further confirmed that the integration of large language models with external tools introduces challenges in planning, selection of arguments, ordering of actions, and degree of reliance on external components.

However, the integration of large language models with external components must always be accompanied by strong constraints on tool use. The model should never select the operating systems to access or the actions to execute in the absence of deterministic controls. For these reasons, this approach to vulnerability prioritization results in a hybrid orchestration mode. The use of deterministic components is appropriate for tasks requiring reproducibility, such as the retrieval of asset metadata, filtering of records, validation of query syntax, and application of fixed rules for policy enforcement. Probabilistic LLM reasoning is appropriate for synthesis, explanation, and natural language justification. The separation of these responsibilities provides additional benefits of flexibility for the overall system while maintaining control over access to data and the implementation of logic for decision-making.

This hybrid approach represents a major component of the overall system described in this study. The resulting orchestration mode represents a boundary between enterprise data sources and large language model-based reasoning methods. It specifies which data should be retrieved, how that data should be reduced, and when the application of the large language model should be used to produce outputs for enterprise analysts. This provides both high levels of efficiency and strong support for the implementation of standards for data access and the logic of security decision-making. In particular, it reduces unnecessary information, improves the capacity for tracing retrieval and reasoning steps, and limits the extent to which probabilistic reasoning contributes to high-risk, high-security workflows.

The technical distinction between deterministic retrieval and probabilistic reasoning serves as a transition to the issue of system governance, which is presented in the subsequent section. RAG, pruning, and orchestration can improve grounding and efficiency, but they do not by themselves make an AI-enabled system acceptable for use in a regulated financial environment. In this regard, the system architecture must also ensure the capability for auditing, control of data boundaries, assignment of accountability, and maintenance of human oversight. For these reasons, the requirements for system governance represent an integral component of the technical design and determine the conditions for the safe implementation of LLM-based reasoning in vulnerability management processes. In this study, RAG is useful only when retrieval is constrained by task-specific workflows, schema-aware joins, and validation checks.

2.5. Governance and Reliability in Financial Systems

The use of AI-supported systems for vulnerability prioritization in financial settings requires not only technical performance but also compliance with requirements for governance, reliability, and operational risk. A system may achieve acceptable performance on a benchmark but remain inappropriate for operational use if it cannot explain its outputs, restrict access to sensitive information, preserve analyst accountability, or conform to existing control procedures. This is especially important for vulnerability prioritization, where inappropriate or incorrect recommendations can impair remediation plans and increase exposure to operational risks.

The requirements for the management of cybersecurity risks, achievement of digital operational resilience, and control of information technology systems further emphasize these concerns [2], [43]. However, they do not specify a particular technical design for AI-based security tools. Instead, they require that such systems be fully documented, well bounded, amenable to audits, and integrated into controlled processes for operational use.

The NIST AI Risk Management Framework identifies trustworthiness characteristics such as validity, reliability, safety, security, transparency, explainability, privacy, and accountability [12]. These characteristics are highly relevant to LLM-based prioritization of vulnerabilities because recommendations affect efforts to achieve remediation and reduce operational risk exposure. For this reason, outputs presented to analysts should not be interpreted as unsupported conclusions but should be subject to review, supported by retrieved information, and consistent with the organization's control environment. Additional guidance for security control and risk management further reinforces the need for documented, controlled, and auditable processes [44], [45], [46].

Broader sources of AI and cybersecurity governance reinforce these limitations. The EU AI Act, NIST Generative AI Profile, NIST Cybersecurity Framework, and EBA guidelines on ICT and security risk management all highlight the requirements for risk management, accountability, security controls, and governance applicable to digital and AI-based systems [18], [47], [48], [49]. In particular, risks associated with prompt injection, leakage of sensitive information, inadequate handling of outputs, and excessive degrees of autonomy are emphasized by the OWASP for applications based on large language models [50].

As a result, the use of a decision-support approach is preferred over an automated strategy for remediation within the context of activities for the management of vulnerabilities. The proposed decision-support framework does not apply patches, change configurations, or initiate remediation actions without human approval. Instead, the system provides analysts with risk information and remediation recommendations while preserving human responsibility for the final decisions. This reduces manual retrieval and interpretation efforts without transferring operational authority to the model.

Table 2.2 summarizes the governance and reliability constraints that influence the design of LLM-based vulnerability prioritization systems in financial environments.

Table 2.2: Governance and reliability constraints for LLM-supported vulnerability prioritization in financial environments.

Constraint	Relevance to vulnerability prioritization	Design implication
Auditability	Analysts must be able to understand why a vulnerability was prioritized	Preserve retrieved evidence and reasoning traces
Data protection	Internal asset and security data may be sensitive	Restrict processing to approved environments and data sources
Reliability	Incorrect recommendations may misdirect remediation effort	Use deterministic retrieval and validation where possible
Accountability	Final operational decisions must remain attributable to responsible staff	Treat the system as decision support, not autonomous remediation
Bounded autonomy	Security workflows should not allow unconstrained model actions	Use orchestration gates and human review before action

These restrictions directly motivate the architecture proposed in this thesis. A generic chatbot interface or loosely constrained tool-using agent would not provide sufficient control over data access, output validation, or the boundary between recommendation and action. A context-aware orchestration framework addresses these concerns by restricting which data sources may be accessed, determining when LLM-based synthesis is appropriate, and validating analyst-facing outputs against retrieved enterprise evidence. Governance requirements are therefore not treated as external constraints added after the system is designed; they shape the architecture itself.

2.6. Research Gap

The literature and standards discussed in this chapter confirm the availability of well-developed but fragmented mechanisms to support vulnerability prioritization. The central research gap addressed in this thesis is the lack of an evaluated framework for integrating context-aware vulnerability prioritization, selective enterprise metadata retrieval, constrained application of LLM-based reasoning, and governance requirements for regulated financial environments. Existing studies provide important components of the overall solution, but do not show how these components can be integrated into a controlled architecture for analyst-facing vulnerability management decision support.

The gap is architectural rather than limited to a single missing metric or model capability. Existing work addresses parts of the problem: vulnerability-prioritization methods describe what information may matter, LLM research shows how models can support security analysis, RAG research explains how external information can be retrieved, and governance frameworks define constraints for operational use. What remains insufficiently developed is an evaluated architecture that combines these elements into a controlled workflow for regulated vulnerability-management decision support. All aspects have been partially addressed in prior studies, but integration across all four dimensions remains inadequate.

The first gap concerns the operationalization of context-aware vulnerability prioritization. Public mechanisms for describing vulnerabilities provide essential reference points for security teams. CVE identifiers make it possible to name publicly known vulnerabilities consistently across scanners, advisories, databases, and remediation workflows [19]. The CVSS provides a standardized method for communicating technical severity [5]. The EPSS adds an estimate of exploitation likelihood [6], whereas the CISA KEV catalogue identifies vulnerabilities for which exploitation has been observed [7]. These mechanisms are valuable because they standardize the description and comparison of vulnerabilities. However, they do not establish the urgency of remediation for a particular asset in a specific organization.

Context-aware methods address some of the limitations of conventional severity-based methods. SSVC focuses on stakeholder-specific decisions for remediation rather than severity scores alone [8], [9]. CAVP provides additional examples of how the inclusion of contextual and temporal information improves methods for vulnerability scoring and prioritization [27]. Recent developments in metrics for vulnerability prioritization also confirm that the use of context-specific, predictive, and aggregated indicators represents a complementary, not interchangeable, source of information [26]. However, the capacity to obtain, filter, normalize, and apply enterprise context in a consistent manner across heterogeneous operational systems remains insufficiently developed.

The second gap concerns the controlled use of LLMs in vulnerability management workflows. Existing cybersecurity benchmarks show that LLMs can support information extraction, summarization, reasoning, and explanations in security tasks. CTIBench evaluates these capabilities for cyber-threat-intelligence tasks [10], whereas SECURE shows that cybersecurity performance should be assessed through domain-specific evaluation rather than inferred from general model ability [31]. CyberSecEval similarly demonstrates the need to evaluate security-relevant model behavior directly [32]. For vulnerability management, these capabilities are useful because analysts must convert technical findings into risk justifications and remediation-oriented communications.

However, the same body of work also shows that LLM outputs may be incomplete, overconfident or insufficiently grounded. Hallucination is a general limitation of LLMs, but it becomes especially problematic in cybersecurity because unsupported claims may influence operational decisions [11]. Work on LLM-based cyber-threat intelligence further shows that errors in extraction and interpretation can occur when the model outputs are not adequately constrained or validated [34]. This risk is particularly important in vulnerability prioritization. A generated statement about exploitability, exposure, ownership, or compensating controls may affect remediation decisions, even when the statement is not supported by enterprise evidence. Therefore, existing work motivates LLM support, but does not justify the use of unconstrained LLMs as independent decision-making agents in high-risk enterprise workflows.

The third gap concerns retrieval and orchestration of data. Retrieval-augmented generation offers a way to ground model outputs in external information rather than relying only on model parameters [14]. However, RAG does not solve the full prioritization problem. Gao et al. show that RAG systems must be evaluated as pipelines because failures can occur during retrieval, augmentation, or generation [15]. Evaluation approaches such as RAGAS and ARES further emphasize that system quality depends on retrieved context relevance, faithfulness to source material, and the quality of the final answer [35], [36]. These findings are directly relevant to enterprise vulnerability data, where relevant evidence may be spread across vulnerability records, asset inventories, ownership data, compliance information and remediation metadata.

The challenge is not only to retrieve information but also to determine which information should be retrieved and how much of it should be passed to the model. Long-context studies have shown that adding more information to a prompt does not guarantee that the model will use the relevant evidence effectively [13]. Adaptive retrieval methods, such as Self-RAG, also support the view that retrieval should be selective rather than indiscriminate [37]. For vulnerability prioritization, this implies that a naive RAG design may retrieve too many records, include irrelevant fields, or obscure the evidence required for a specific decision. The missing element is a task-specific orchestration mechanism that controls data source selection, metadata pruning, validation, and the boundary between deterministic processing and probabilistic synthesis.

The fourth gap concerns governance in regulated financial infrastructure. DORA and NIS2 establish expectations for digital operational resilience, cybersecurity risk management, and control over ICT systems [2], [43]. The EU AI Act and NIST AI Risk Management Framework add broader requirements for accountability, transparency, reliability, and risk management in AI-enabled systems [12], [47]. The NIST Generative AI Profile and OWASP guidance on LLM applications further highlight risks such as prompt injection, sensitive data exposure, inadequate output handling, and excessive autonomy [48], [50]. These sources do not prescribe a specific architecture for the LLM-supported vulnerability prioritization. However, they imply that systems used in financial security operations must be bounded, auditable, reviewable, and compatible with human accountability.

Together, these limitations demonstrate that the problem is not simply a lack of another scoring method, benchmark, or chatbot interface. The missing contribution is an evaluated architecture that integrates context-aware prioritization, selective access to enterprise metadata, constrained LLM-based synthesis and governance-compatible decision support. Existing studies provide important individual components but do not show how these components can be combined into a controlled orchestration framework for regulated vulnerability-prioritization workflows.

This thesis addresses the gap by developing, implementing, and evaluating an integrated context-aware orchestration framework for vulnerability prioritization in financial infrastructure. The contribution is both architectural and evaluative: the thesis defines how deterministic retrieval, metadata pruning, validation, and bounded LLM-based synthesis can be combined, and compares this strategy with a less constrained autonomous-agent baseline on representative benchmark tasks. This evaluation provides evidence about the conditions under which LLM-supported prioritization can remain grounded, traceable, and operationally useful in a regulated setting.

3
Requirements and Evaluation Approach

This chapter describes the requirements and evaluation methodology applied to develop and evaluate the proposed framework. First, it describes the functional and non-functional requirements based on the problem context. Second, it details the benchmark environment, test cases, metrics, and methods of testing for robustness and financial impact, and uses them to evaluate and compare orchestration paradigms.

This thesis adopts a design-science research orientation. It addresses a practically relevant problem in vulnerability prioritization, develops an artifact in the form of a context-aware orchestration framework, and validates the artifact using representative benchmark tasks and operational metrics. This constitutes compliance with the principles of design-science research, in which artifact usefulness is achieved through validation in the context of a well-defined problem [51], [52], [53].

3.1. Framework Requirements

The framework requirements reflect the need to enable vulnerability prioritization in a regulated financial environment. The resulting system must integrate public information about vulnerabilities with internal information about the organizational context and maintain traceability, operational efficiency, and control by the information security analysts. The requirements are categorized into functional requirements, which define the capabilities that must be implemented, and non-functional requirements, which specify the quality attributes that must be achieved.

3.1.1. Functional Requirements

The functional requirements below translate the framework objective into concrete capabilities for retrieving the enterprise context, supporting prioritization decisions, and producing analyst-facing outputs.

FR1: Heterogeneous data retrieval. The framework must retrieve and combine data from heterogeneous sources, including asset metadata, vulnerability intelligence, compliance information, ownership records, exposure indicators, and remediation contexts.

FR2: Context-aware prioritization. The framework must support prioritization decisions by correlating public vulnerability signals, such as CVSS and EPSS, with asset-specific context, such as exposure, ownership, operational role, compliance status, and compensating controls.

FR3: Analyst-facing justification. The framework must generate natural language explanations that justify prioritization decisions using the retrieved enterprise context. These explanations should be suitable for security analysts and should distinguish between retrieved evidence and generated synthesis.

FR4: Remediation-oriented support. The framework must support remediation guidance by identifying affected assets, relevant owners, urgency levels, and operational remediation contexts, such as maintenance windows or patching constraints.

FR5: Deterministic data processing. The retrieval and structuring of factual enterprise data must be handled through programmatic orchestration and predefined schemas. The framework should not rely on the LLM to autonomously discover database structures, infer schemas, or decide which factual records exist in the database.

FR6: Multi-use-case support. The framework must support multiple vulnerability management tasks, including initial triage, risk justification, and remediation-oriented reporting.

3.1.2. Non-Functional Requirements

While the functional requirements define what the framework must do, the non-functional requirements define how those capabilities must be delivered safely, reliably, and transparently in a regulated financial environment.

NFR1: Auditability and traceability. Analyst-facing recommendations must be traceable to the internal records and vulnerability information used for their generation. This requirement is necessary because security decisions in regulated financial environments must be explicable and reviewable.

NFR2: Reliability. The framework must reduce unsupported claims and minimize errors in high-impact fields, such as asset exposure, ownership, and compliance status. Reliability is prioritized over autonomy because incorrect prioritization may misdirect remediation efforts.

NFR3: Operational efficiency. The framework must limit redundant tool calls and unnecessary token consumption so that the approach remains practical at an enterprise scale.

NFR4: Robustness. The framework must degrade gracefully when exposed to missing fields, incorrect parameters, dropped records, hallucinated lines or incomplete tool-call chains.

NFR5: Data boundary control. The framework must minimize the exposure of sensitive internal metadata by ensuring that only the necessary context is passed to the LLM-based synthesis component and that processing remains within approved environments.

NFR6: Human oversight. The framework must operate as a decision support system rather than an autonomous remediation system. The final prioritization and remediation decisions remain the responsibility of human analysts.

Table 3.1 summarizes how the requirements are related to the research questions.

Table 3.1: Summary of framework requirements and their relation to the research questions.

Requirement	Purpose	Related RQ
FR1, FR5	Retrieve and structure heterogeneous enterprise context through controlled orchestration	RQ1
FR2, FR3, FR4	Support context-aware triage, justification, and remediation tasks	RQ1, RQ2
NFR1, NFR5, NFR6	Preserve traceability, data boundaries, and analyst oversight	RQ3
NFR2, NFR3, NFR4	Evaluate accuracy, efficiency, and robustness under realistic workflow constraints	RQ2, RQ3

3.2. Evaluation Design

The evaluation compares two orchestration paradigms for LLM-supported vulnerability management tasks in a controlled offline banking environment. The first paradigm is a parametric orchestration framework that follows structured workflow templates and uses restricted tool access. The second paradigm is an autonomous agent with substantially greater access to tools and the freedom to determine its own order of operations and reasoning strategy. This comparison was intended to test whether constrained orchestration improves the accuracy, efficiency, and robustness of well-defined vulnerability management workflows.

The comparison was structured as a controlled evaluation, with both orchestration paradigms evaluated using the same data sources, benchmark questions, scoring rubric, and execution conditions. This follows the common principles of empirical software engineering experimentation, where alternative techniques should be compared under controlled and repeatable conditions [54].

3.2.1. Benchmark Environment

The benchmark was implemented in a controlled proof-of-concept environment using an authorized offline deployment of the Claude Opus 4.6 model for LLM-based reasoning. Task-specific prompt templates and output format requirements enabled the evaluation of targeted vulnerability management applications. The use of well-structured contextual information obtained by deterministic retrieval ensured reproducibility. To achieve maximum consistency, identical datasets, prompt templates, specifications for complete workflow operations, and seeding of random-number generation were applied to all experiments.

The benchmark data include three approved static production-derived comma-separated values (CSV) snapshot sources. The vulnerability data represent improved records of exploitation with evidence of compromise, the application of MITRE ATT&CK techniques, and quantification of exploit maturity. The asset data represent a simplified ServiceNow-style configuration management export that includes production and staging servers with information on the owner, operating system, software, and location. The resulting records of compliance and risk represent the context required to evaluate the use cases and are valid only at the time of the snapshot.

This design preserves all relationships required to evaluate orchestration behavior and prevents interaction between the framework and LLM components with live enterprise systems. Sensitive data were partially masked to maintain confidentiality. This modification distorted names and modes of representation but preserved all relationships required for evaluation, including associations among vulnerabilities, affected assets, ownership contexts, and compliance records. Therefore, the benchmark reflects actual configurations and records for selected application cases; however, the results must be interpreted as representing controlled, offline results of proof-of-concept experiments and not as representing actual experiences with live production systems.

Table 3.2: Approved static production-derived CSV snapshot sources used in the benchmark environment.

Data source	Records	Description
Vulnerability snapshot	5 CVEs	Production-derived vulnerability records enriched with indicators of compromise, MITRE ATT&CK information, and exploit maturity
CMDB snapshot	6 servers	Production-derived asset metadata for selected production and staging servers, including owner, operating system, and installed software
NSF compliance snapshot	6 records	Production-derived per-asset compliance scores and control-level findings for the selected assets

This benchmark setting was deliberately controlled to isolate the orchestration behavior of the two paradigms. The static snapshot preserved the relationships required for evaluation, including associations between vulnerabilities, affected assets, ownership context, and compliance records, while preventing interaction with live enterprise systems. The benchmark therefore supports a repeatable proof-of-concept comparison, but it should not be interpreted as a measurement of performance in a live banking environment.

3.2.2. Use Cases and Benchmark Tasks

The benchmark consists of 14 questions grouped into three representative use cases. Each use case corresponds to a different type of analytical workflow.

UC1: Contextual asset triage. The system must retrieve affected assets for a given vulnerability and combine this information with ownership and compliance contexts. This use case evaluates multi-hop retrieval and aggregation across vulnerability, configuration management database (CMDB), and compliance sources.

UC2: Risk-score justification. The system must combine generic vulnerability severity with internal asset classification and environmental context to produce a justified priority rating. This use case evaluates context-aware reasoning and explanations.

UC3: Remediation guidance and query generation. The system must generate remediation-oriented outputs, including urgency tiers, affected hosts, and threat-hunting query support. This use case evaluates the ability to produce actionable, analyst-facing guidance.

These three use cases reflect the major operational phases of analyst-facing vulnerability prioritization. Contextual asset triage identifies affected assets, owners, and compliance contexts. Risk score justification describes how general measures of vulnerability severity should be interpreted or modified in the context of internal assets and environmental factors. Finally, remediation guidance translates the resulting priorities into actionable support for patching teams and threat-hunting activities. Together, these activities implement the functions of retrieval, contextual interpretation, and operational communication, which represent the essential capabilities required for LLM-based vulnerability prioritization in this setting.

Table 3.3: Benchmark use cases and evaluated capabilities.

Use case	Questions	Evaluated capability
UC1	Q1–Q6	Multi-hop retrieval from CVE, CMDB, ownership, and compliance sources
UC2	Q7–Q10	Context-aware risk justification and priority explanation
UC3	Q11–Q14	Remediation guidance and Microsoft Sentinel KQL query generation

3.2.3. Evaluation Metrics

Accuracy was measured as weighted task correctness against predefined ground-truth elements. For each benchmark question, the expected answer specified the affected servers, responsible owners, and supporting fields required for a complete response. The final score therefore reflects whether the system retrieved and reported the evidence required for the task, rather than whether the generated prose was stylistically convincing.

The weighted score for a task is defined as:

$$W_{\text{total}} = 0.35 S_{\text{server}} + 0.35 S_{\text{owner}} + 0.30 S_{\text{field}}. \tag{3.1}$$

Here, S_{server} denotes the fraction of expected affected servers correctly identified, S_{owner} denotes the fraction of expected owners correctly identified, and S_{field} denotes the fraction of expected supporting fields included in the answer. The weights give slightly greater importance to affected asset and owner identification because these elements determine whether remediation can be assigned to the correct operational team. Supporting fields include values such as compliance score, risk level, patch status, or remediation context, depending on the task.

The current evaluation does not directly quantify analyst workload via user studies, measures of time-on-task, or subjective estimates of workload. Rather, it quantifies operational overhead with surrogate metrics of the number of tool calls, prompt tokens, completion tokens, total token costs, response latencies, and predicted costs of error correction for incomplete or incorrect results. These data provide estimates of the operational costs associated with each orchestration paradigm but should not be regarded as direct measures of analyst workload.

In addition to accuracy, the evaluation also captures operational efficiency metrics such as total tool calls, prompt tokens, completion tokens, total tokens, and latency. These results enable a comparison of the operational costs of parametric and autonomous orchestration paradigms. The full benchmark question set, expected ground truth fields, and per-question scores are provided in Appendix A.

3.2.4. Compared Orchestration Paradigms

The evaluation compares two orchestration paradigms: a constrained parametric framework and a less-constrained autonomous agent baseline. The purpose of this comparison is to isolate the effect of orchestration control on accuracy, operational overhead, robustness, and estimated correction effort.

Parametric orchestration. The parametric approach uses predefined workflow templates for each use case. Tool access is restricted to the subset required for the task, and the order of factual retrieval steps is specified programmatically. The LLM is primarily used for synthesis after the relevant data are retrieved, structured, and validated.

Autonomous orchestration. The autonomous approach provides the LLM with broader access to the available toolset and allows the model to decide the order and strategy of tool use. This approach represents a less constrained agentic baseline in which the model is responsible for planning its own retrieval and reasoning processes.

Table 3.4: Compared orchestration paradigms in the benchmark.

Dimension	Parametric orchestration	Autonomous orchestration
Tool access	Restricted to required tools	Full toolset available
Workflow	Fixed multi-step templates per use case	LLM decides tool order and strategy
Retrieval control	Programmatic and schema-driven	Model-planned and less constrained
Main advantage	Traceability and reliability for known workflows	Flexibility for exploratory or low-structure tasks
Main risk	Lower flexibility outside predefined workflows	Skipped hops, redundant calls, and unsupported synthesis

To isolate the effects of orchestration strategy, all other experimental conditions were kept constant to the extent possible. Both paradigms were tested on identical benchmark questions, the same approved static production-derived CSV snapshot, identical deterministic tool backends, identical prompt-template families, identical output format expectations, identical decoding temperatures, identical maximum iteration limits, identical validation rules, identical scoring rubrics, identical stopping criteria, and identical offline model configurations. As a result, the only independent variable was the level of control over orchestration: fixed task-specific workflow templates in the parametric approach versus model-planned tool use in the autonomous agent baseline.

Table 3.5: Controlled and intentionally varied conditions in the baseline comparison.

Condition	Parametric framework	Autonomous baseline	Status
S Model deployment	Same offline model configuration	Same offline model configuration	Controlled
Benchmark data	Same approved static production-derived CSV snapshot	Same approved static production-derived CSV snapshot	Controlled
Tool backend	Same deterministic tools	Same deterministic tools	Controlled
Prompt conditions	Same task family and output-format expectations	Same task family and output-format expectations	Controlled
Decoding temperature	Same	Same	Controlled
Iteration limits	Same maximum limits	Same maximum limits	Controlled
Validation rules	Same	Same	Controlled
Stopping criteria	Same	Same	Controlled
Scoring rubric	Same weighted scoring method	Same weighted scoring method	Controlled
Workflow control	Fixed task-specific templates	Model-planned sequence	Independent variable
Tool-selection freedom	Restricted to required tools	Broader tool access	Independent variable

The methods deliberately vary with respect to the freedom of tool selection and degree of workflow planning. These variations do not represent uncontrolled confounding factors but instead reflect the different orchestration paradigms under comparison. Therefore, the comparison evaluates whether restricting the retrieval order, tool access, and validation around the LLM improves performance and reliability for the selected controlled benchmark tasks.

This comparison directly supports RQ2 and RQ3. RQ2 is addressed by measuring accuracy, operational overhead, estimated correction effort, and operational cost for both paradigms. RQ3 is addressed by analyzing whether constrained orchestration provides safeguards relevant to regulated financial environments, including traceability, data-boundary control, validation, and human oversight.

3.3. Robustness Testing

The robustness evaluation assesses how the two orchestration paradigms respond to controlled stochastic perturbations of workflow execution. The goal is not to estimate real production failure rates, but to test whether each paradigm remains reliable when retrieval, tool use, or output generation is imperfect. This distinction is important because the perturbations model orchestration-level failures, such as omitted retrieval steps or incomplete integration of retrieved records, rather than failures of the underlying data sources. Section 3.3.1 defines the stochastic noise model, while Section 3.3.2 describes the sensitivity analysis used to test whether the robustness results depend on a single noise calibration or random seed.

3.3.1. Stochastic Noise Model

A noise-free benchmark would enable perfect accuracy for both orchestration paradigms in the controlled snapshot benchmark. However, this scenario does not represent the error conditions that arise in workflows supported by large language models. To assess robustness, the benchmark incorporates a stochastic noise model that mimics the errors of tool use and synthesis. The application of explicit perturbations facilitates a more rigorous characterization of failure behavior and allows the separation of performance under optimal conditions from performance under conditions of imposed error [54], [55].

The noise configuration is represented as a six-tuple:

$$\mathcal{N} = (p_{\text{hop}}, p_{\text{drop}}, p_{\text{param}}, p_{\text{hall}}, p_{\text{extra}}, p_{\text{field}}). \tag{3.2}$$

Each parameter p_i represents the Bernoulli probability that a specific error type is injected at an applicable point during a single-agent run. The six error types are missed tool-call hops, dropped server records, incorrect tool parameters, hallucinated report lines, extra tool calls, and missed output fields.

These probabilities are modeling parameters rather than empirical estimates of production failure rates. Rather, they are parameters for modeling errors that arise from the orchestration of component services. For example, the loss of a server record reflects the failure to aggregate or incorporate that record in the final response, not the failure of the CMDB snapshot itself. This difference is important because both orchestration approaches employ identical interfaces for the same underlying data. As a result, the increased probabilities of autonomous events represent errors attributable to the planned selection of tools, construction of arguments, management of intermediate states, and integration of final results.

The probabilities differ between the two paradigms because the injected errors reflect patterns of workflow performance and not raw measures of tool reliability. The parametric approach limits the sequence of operations, expected operands, modes of aggregation, and forms of output, whereas the autonomous mode of operation requires dynamic planning for these functions. Therefore, greater probabilities of error were assigned to the autonomous mode for errors in planning, operand selection, management of intermediate states, and integration of final results.

Table 3.6: Error types in the stochastic noise model.

Symbol	Error type	Effect
p_{hop}	Missed hop	A required tool-call step is skipped, causing the absence of downstream data.
p_{drop}	Dropped server	One expected server is lost during aggregation, context construction, or final response synthesis.
p_{param}	Wrong parameter	A tool call is issued with an incorrect, incomplete, or malformed argument.
p_{hall}	Hallucinated line	A fabricated but plausible-looking line is added to the final response.
p_{extra}	Extra tool call	An unnecessary tool call is inserted into the workflow.
p_{field}	Missed field	An expected output field or data section is omitted from the constructed context or the final response.

The noise profiles differ between the two orchestration approaches. The parametric approach reduces opportunities for errors of missed hops, additional tool calls, incorrect parameters, and fabricated lines and thus assigns lower probabilities to these errors. In contrast, the autonomous approach provides greater freedom for the agent to generate tool order, arguments, and final synthesis, resulting in higher probabilities for the same error types.

Table 3.7: Configured stochastic noise profiles for the two orchestration paradigms.

Error type	Symbol	Parametric profile	Autonomous profile
Missed hop	p_{hop}	0.06	0.22
Dropped server	p_{drop}	0.10	0.18
Wrong parameter	p_{param}	0.04	0.12
Hallucinated line	p_{hall}	0.05	0.15
Extra tool call	p_{extra}	0.02	0.25
Missed field	p_{field}	0.08	0.14

At each injection point, the decision to inject an error is modeled as an independent Bernoulli trial:

$$\text{inject}_i = \begin{cases} 1, & \text{if } U_i < p_i, \\ 0, & \text{otherwise,} \end{cases} \quad U_i \sim \text{Uniform}(0,1). \tag{3.3}$$

Here, U_i is a uniformly distributed random variable and p_i is the probability of injecting error type i. Although independent of one another, the effects of injected errors may be functionally interrelated. For example, the loss of a server record at the time of CMDB retrieval also results in the loss of related owner and compliance information in the final response. This leads to cascading reductions in the scores for the server, owner, and field dimensions.

The purpose of the noise model was to determine whether structured orchestration degraded more gracefully than fully autonomous orchestration in the presence of realistic workflow errors. This test is especially important for financial applications, where the lack of complete or supported information about security events can result in high costs for manual review or inappropriate prioritization of corrective actions.

3.3.2. Sensitivity Analysis Design

To minimize the risk that the robustness result depended on a single calibration of the stochastic noise model, a sensitivity analysis was conducted using three global scaling factors for noise and five random-number seeds. The scaling factors represented optimistic, nominal, and pessimistic error conditions of 0.5$\times$, 1.0$\times$, and 1.5$\times$, respectively. For each scaling factor, all six Bernoulli probabilities for the parametric and autonomous noise profiles were multiplied by the scaling factor and restricted to the interval $[0, 1]$. The random seeds were 7, 42, 123, 256, and 999, respectively.

The objective of this analysis was not to quantify production failure rates but to assess whether the relative robustness of orchestration paradigms is maintained in the face of varying assumptions about the magnitude of noise and the random nature of injected errors. The results of this sensitivity analysis are presented in Section 6.5.

3.4. Financial Impact Modeling

The analysis also includes the operational costs of each orchestration paradigm. Direct costs are derived from token consumption, whereas indirect costs are modeled as estimated human review or correction costs associated with incomplete or incorrect responses. This separation is important because the direct costs of model usage may be relatively small compared to the estimated cost of reviewing or correcting incomplete outputs. Overall, this approach is consistent with the principles of information security economics [56], [57], [58] that stress the importance of combining technical controls with considerations of incentives, effort, and cost to reduce expected losses.

The per-query token cost is computed from the number of prompt and completion tokens as follows:

$$C_q = \frac{T_{\text{in}}}{10^6} P_{\text{in}} + \frac{T_{\text{out}}}{10^6} P_{\text{out}}, \tag{3.4}$$

where T_{in} is the number of input tokens, T_{out} is the number of output tokens, P_{in} is the price per million input tokens, and P_{out} is the price per million output tokens.

The annual direct token cost can then be estimated as:

$$C_{\text{annual}} = C_q \cdot Q_{\text{day}} \cdot 365, \tag{3.5}$$

where Q_{day} is the number of daily analyst queries.

The human review cost is modeled separately because inaccurate or incomplete outputs may require analyst correction. The annual human review cost is estimated as follows:

$$H_{\text{annual}} = Q_{\text{day}} \cdot r_{\text{review}} \cdot C_{\text{review}} \cdot 365, \tag{3.6}$$

where r_{review} is the fraction of outputs that require manual review, and C_{review} is the estimated cost of a single manual review. This model is used in Chapter 6 to analyze whether improvements in accuracy and robustness produce operational cost savings beyond direct token reductions.

An output is considered to require a manual review when it is incomplete, contains unsupported generated information, omits required fields for ownership or assets, or does not conform to the expected output structure for the task. The review rate parameter represents an operational assumption based on patterns of observed errors rather than a measurement of the actual rate of production SOC review.

4 Framework Architecture

This chapter describes the architecture of the proposed context-aware orchestration framework. The architecture realizes the requirements defined in Chapter 3 by separating deterministic retrieval from LLM-based synthesis. It discusses the architectural objectives, overall components, data model, orchestration layer, decision gate, retrieval and pruning mechanisms, synthesis path, validation methods, and trust boundaries.

4.1. Architectural Objectives

The architecture is based on the functional and non-functional requirements established in Chapter 3. The first objective is to support vulnerability prioritization by integrating public vulnerability information with enterprise-specific context while maintaining control over data processing. The framework therefore combines vulnerability intelligence, asset metadata, ownership information, exposure indicators, and compliance records into task-specific evidence for analyst-facing decision support.

The second objective is to avoid assigning factual discovery to the LLM. The model should not infer database schemas, determine which enterprise records exist, or establish relationships among operational systems. These activities are performed by deterministic components that implement predefined retrieval steps and schema-based joins. As a result, the factual basis of each response can be inspected independently from the natural-language explanation presented to the analyst.

The third objective is to reduce the amount of enterprise data provided to the LLM. Operational records often contain fields that are irrelevant to a specific prioritization task. Passing complete records to the model would increase token costs, expose unnecessary internal information, and make the generated output harder to validate. The architecture therefore includes a context-construction stage that retains only the fields required for the selected workflow.

The fourth objective is to preserve analyst accountability. The framework supports prioritization, risk justification, and remediation guidance, but it does not apply patches, modify configurations, or make final operational commitments. This boundary is especially important in regulated financial environments, where recommendations must remain subject to human review and operational accountability.

These objectives result in a hybrid architecture in which deterministic components perform retrieval, joining, pruning, and validation, while the LLM is used for bounded synthesis and analyst-facing explanation. The rest of this chapter describes how these components interact.

4.2. High-Level Architecture

Figure 4.1 presents the high-level architecture of the proposed framework. The architecture defines explicit boundaries between deterministic control and probabilistic reasoning. This distinction is central to the design because vulnerability prioritization depends on factual records of affected assets, ownership, exposure, and compliance status. Incorrect selection or integration of these records would result in explanations that appear valid but are not supported by the underlying data.

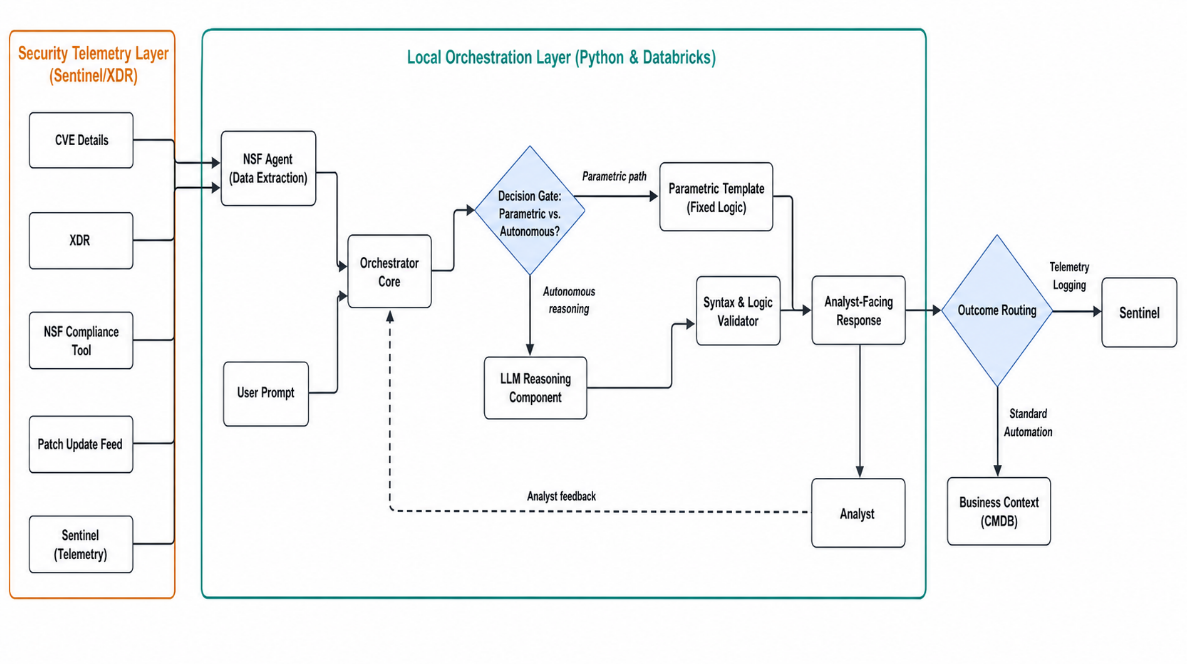

Figure 4.1: High-level architecture of the context-aware orchestration framework. Analyst queries are processed using an orchestration layer that selects the appropriate workflow and routes the request through a decision gate. Factual enterprise data are retrieved through deterministic components, reduced through metadata pruning, and supplied to the LLM-based synthesis component only after normalization. The generated output is validated and presented as analyst-facing decision support rather than autonomous remediation.

The figure should be read from left to right. An analyst query first enters the orchestration layer, where the query type is identified and mapped to an appropriate workflow. The decision gate then determines whether the request can be handled through deterministic retrieval, whether it requires LLM-based synthesis after retrieval, or whether it should follow a hybrid workflow that combines retrieval, context construction, synthesis, and validation. Enterprise data sources are not exposed directly to the LLM. Instead, vulnerability records, CMDB records, ownership information, exposure indicators, and compliance records are accessed through predefined retrieval functions and schema-based joins.

The parametric path is used when the required retrieval sequence can be specified in advance. In this path, deterministic components retrieve, join, prune, and normalize the relevant records before any LLM-based synthesis takes place. The resulting records are transformed into compact, task-specific context objects that contain only the fields required for the selected prioritization task. This design reduces token consumption, limits unnecessary disclosure of internal operational information, and improves the traceability of the factual basis used in the final response.

The LLM-based synthesis component is used only after the relevant enterprise context has been assembled. Its role is therefore limited to explanation, summarization, risk justification, or remediation-oriented communication, rather than factual discovery. Subsequent validation and output-formatting modules check whether the generated response is complete, well-organized, and consistent with the retrieved information before it is returned to the analyst.

This design also defines the main architectural contrast with the autonomous-agent baseline evaluated later in this thesis. In an autonomous-agent design, the LLM may select which tools to invoke, in what sequence, and with what intermediate representation. In the proposed framework, these choices are constrained by predefined workflows that increase inspectability and support the financial-governance requirements described in Section 2.5. These requirements include auditability, control of data-flow boundaries, and human oversight. The implications of this architectural distinction are evaluated in Chapter 6, where the parametric orchestration approach is compared with the autonomous-agent baseline in terms of accuracy, overhead, robustness, and estimated correction effort.

4.3. Data Model and Retrieval Relationships

The proof-of-concept data model supports three logical domains that are sufficient for evaluating the context-aware prioritization workflows used in this study. Vulnerability intelligence characterizes the technical properties of vulnerabilities, asset metadata describes where those vulnerabilities occur within the enterprise, and compliance metadata provides information about control status and residual risk for affected assets.

The vulnerability-intelligence domain is implemented with CVE-based records. These records include vulnerability descriptions, severity levels, CVSS scores, affected software, patch availability, exploit maturity, indicators of compromise, and associated MITRE ATT&CK techniques. This information supports both prioritization and remediation-oriented guidance.

Asset metadata are represented through records in a ServiceNow-style CMDB. Each asset record contains an asset identifier, operating system, owner, support group, environment, classification, installed software, and associated vulnerabilities. The relationship between vulnerabilities and affected assets is captured through the `cves_affecting` field.

The compliance domain is represented through NSF records that describe compliance status, control findings, overall compliance score, risk level, and remediation dates. These records are linked to CMDB assets through the `sys_id`/`asset_id` relationship. This relationship provides the basis for deterministic retrieval of compliance and risk information for affected assets.

Figure 4.2 summarizes the retrieval relationships among the three logical data domains.

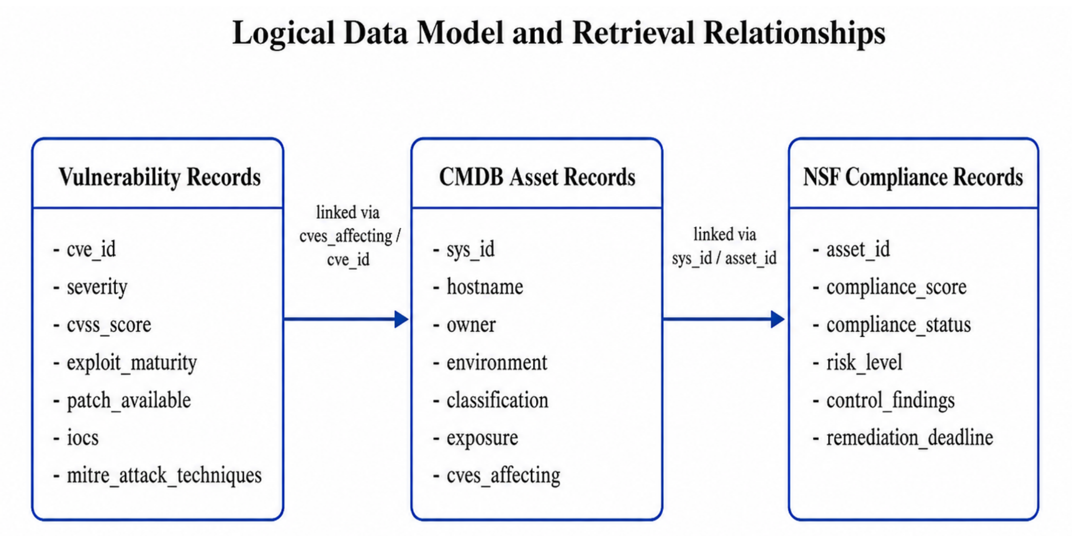

Figure 4.2: Logical data model used by the framework. Vulnerability records are linked to affected assets through the CMDB field `cves_affecting`. Asset records are linked to compliance records through the `sys_id`/`asset_id` relationship. These joins enable deterministic multi-hop retrieval from vulnerability intelligence to asset ownership, classification, exposure, and compliance context.

The data model supports the three benchmark use cases described in Section 3.2.2. In contextual asset triage, the workflow begins with a CVE identifier and identifies affected assets through the `cves_affecting` relationship. These assets are then enriched with ownership and compliance context. In risk-score justification, vulnerability severity and exploitability are combined with asset classification, environment, exposure, and compliance risk. In remediation guidance, patch availability, affected systems, ownership requirements, remediation deadlines, indicators of compromise, and relevant ATT&CK techniques are used to construct an operational summary.

The purpose of the data model is not to reproduce the full complexity of a production banking environment. Instead, it captures the minimum set of relationships required to evaluate the orchestration problem: vulnerability records must be linked to affected assets, and asset records must be enriched with ownership and compliance context. These relationships provide the basis for the deterministic retrieval path described in the next section.

4.4. Orchestration Layer

The orchestration layer represents the central control component of the framework. It coordinates retrieval, context construction, LLM-based synthesis, and validation by mapping each analyst query to a supported workflow. In doing so, it controls which operations are available for each task and determines the order in which they are applied. For this proof-of-concept, the supported workflows correspond to the three use cases introduced in Section 3.2.2: contextual asset triage, risk-score justification, and remediation guidance.

Each workflow defines a fixed pattern for retrieving and integrating evidence. For example, a contextual asset-triage query begins with a vulnerability identifier, retrieves the corresponding vulnerability record, identifies affected assets, adds ownership and compliance context, and returns a structured analyst-facing output. The purpose of this workflow is not only to obtain the relevant records, but also to ensure that the required evidence chain is constructed in a predictable order.

This design differs from an autonomous-agent architecture, where the LLM may decide which tools to invoke, which arguments to use, and whether additional retrieval steps are necessary. Such flexibility can be useful for exploratory tasks, but it also introduces opportunities for omitted retrieval steps, redundant tool use, and inconsistent intermediate states. In the proposed framework, tool use is instead constrained by predefined workflow templates. This allows the framework to preserve traceability, reduce unnecessary exposure of enterprise data, and ensure that LLM-based synthesis is performed only after the task-specific context has been assembled.

Table 4.1: Responsibilities of the orchestration layer.

Responsibility	Description
Workflow selection	Maps the analyst query to a supported use case, such as triage, risk justification, or remediation guidance.
Tool control	Restricts tool use to the operations required by the selected workflow.
Retrieval coordination	Executes deterministic retrieval steps across vulnerability, asset, ownership, and compliance data.
Context construction	Converts raw records into a compact task-specific context object.
LLM boundary enforcement	Ensures that the LLM receives only selected context rather than unrestricted access to enterprise data.
Trace preservation	Stores intermediate retrieval results and joins so that analyst-facing outputs can be inspected.

4.5. Decision Gate

The decision gate implements a routing function within the orchestration layer and determines the processing required prior to data retrieval or LLM-based reasoning. Incoming requests are categorized based on the task type, required data sources, and required depth of reasoning. The decision gate then invokes a deterministic, LLM-supported, or hybrid processing pathway. Finally, a validation pathway is used to ensure that the output conforms to the requirements for schema, syntax, or grounding constraints.

The use of a deterministic pathway indicates that the requested output can be achieved by structured retrieval and formatting alone. Examples include listing assets affected by a known CVE, providing ownership information for a server, or reporting the status of compliance for an asset. These tasks require accuracy of fact and schema-based integration of data but do not involve open-ended LLM-based reasoning. The application of LLM-based processing at this stage would increase costs and risks without improving the accuracy of the results.

The use of an LLM-assisted pathway is appropriate for requests that require the synthesis or interpretation of information obtained by factual data retrieval. Examples include an explanation of the basis for prioritizing a vulnerability, integration of information on the urgency of remediation for multiple assets, or generation of a report for use by technical personnel. In such instances, the components of the deterministic pathway retrieve and organize all the necessary information for the subsequent LLM-based generation of an interpretable summary of the results.

The application of a hybrid pathway is appropriate for requests that require both deterministic computation and LLM-based interpretation. For example, information on the availability of software patches, affected hosts, asset ownership, compliance testing results, and indicators of compromise would be retrieved deterministically. An LLM would then be used to produce an integrated summary of the information or to generate a template for a query directed toward specialists in threat hunting.

The overall sequence of events ensures that all factual retrieval operations are completed before LLM-based synthesis is applied. This approach is deliberately conservative. For requests involving sensitive information on operational status or requiring precise selection of fields of factual data, preference is given to completing and validating all operations of deterministic data retrieval and avoiding the application of unconstrained LLM-based reasoning. This reflects the role of the system as a tool for data-based analytical decision support. The system is intended to provide a basis for analytical decisions by generating well-documented and highly structured representations of data and not to make independent remediation decisions.

Table 4.2: Decision-gate routing rules.

Route	Trigger condition	Example task
Deterministic retrieval	The query asks for factual records or direct joins between known entities.	Identify the servers affected by a CVE and retrieve their owners.
LLM-assisted synthesis	The query asks for an explanation, summarization, or analysis-facing justification based on the retrieved context.	Explain why a vulnerability should be treated as a high priority.
Hybrid workflow	The query requires deterministic retrieval, followed by structured synthesis or report generation.	Generate remediation guidance for affected hosts using patches and compliance contexts.
Validation path	The query produces a structured output that must satisfy a schema or syntax constraint.	Validate the required fields or check the generated KQL structure before returning the output.

Figure 4.3 illustrates the role of the decision gate in routing analyst queries through the appropriate processing paths.

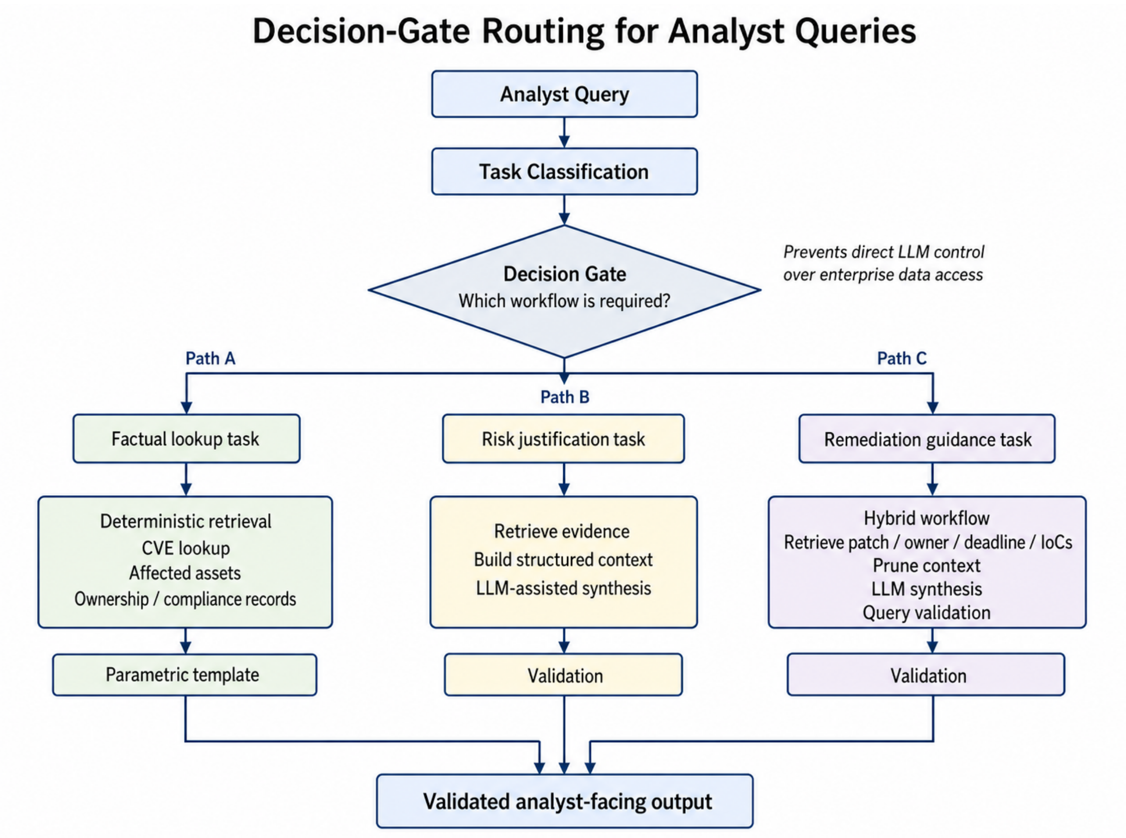

Figure 4.3: Decision-gate routing in the proposed framework. Analyst queries are first classified according to task type and required synthesis depth. Factual lookup tasks follow a deterministic retrieval path, explanation tasks use LLM-assisted synthesis after retrieval, and remediation tasks use a hybrid workflow that combines retrieval, pruning, synthesis, and validation. This routing prevents the LLM from directly controlling enterprise data access.

The decision gate represents a key distinction between this approach and that of a fully autonomous agent. In the latter case, the model may select among various actions, including inspection of schemas, invocation of tools, omission of intermediate steps, and generation of a response based on partial information. In this architecture, these choices are restricted by predefined routing rules to produce a more predictable workflow that facilitates traceability and minimizes the loss of important enterprise context.

4.6. Deterministic Retrieval Path

The deterministic retrieval pathway extracts factual information from available data sources before any LLM-based inference. This method is deterministic because the order of retrieval operations is dictated by the chosen workflow and not by the arbitrary decisions of the language model. As a result, the factual information used to produce results for analysts can be traced unambiguously to specific records and joins. This approach is applicable to the three logical data domains described in Section 4.3: vulnerability intelligence, asset metadata, and data on compliance with information security policies.

The proof-of-concept implements this path over three static CSV-based data domains: vulnerability, CMDB asset, and NSF compliance records. The retrieval sequence follows the relationships introduced in Section 4.3. A CVE identifier is first used to retrieve the relevant vulnerability records. The CMDB snapshot is then queried to identify the assets affected by the CVE through the `cves_affecting` field. Finally, each affected asset is linked to the NSF compliance records through the `sys_id`/`asset_id` relationship.

The retrieved fields depend on the selected use case. For contextual asset triage, the path identifies affected assets, ownership details, operating environment, classification status, compliance posture, and risk indicators. To justify the risk scores, the path integrates measures of vulnerability severity and exploit maturity with asset-specific and compliance-based risk factors. Finally, it incorporates operationally relevant information on the availability of patches, deadlines for remediation, responsibility for support, signs of compromise, and use of MITRE ATT&CK techniques.

This approach eliminates repeated ad hoc retrievals by the LLM. The model is not required to determine which records should be integrated or which fields are appropriate for a specific task. Instead, the deterministic pathway provides a complete and task-specific factual context prior to any synthesis by a language model.

Table 4.3 summarizes the retrieval paths for each use case.

Table 4.3: Deterministic retrieval path per benchmark use case.

Use case	Retrieval sequence	Main retrieved fields
UC1: Contextual asset triage	CVE → CMDB assets → NSF compliance	Affected servers, owners, environment, classification, compliance status, control findings
UC2: Risk-score justification	CVE → CMDB asset context → NSF risk context	Severity, CVSS score, exploit maturity, asset classification, environment, compliance score, risk level
UC3: Remediation guidance	CVE → affected assets → patch and compliance context	Patch availability, patch version, owners, remediation deadline, IoCs, MITRE ATT&CK techniques

This deterministic path provides two advantages. First, it increases reliability by explicitly encoding the necessary steps for information retrieval within the workflow. Second, it increases auditability because each output can be traced directly to the source records used for its generation. As a result, the model is not responsible for identifying factual relationships among the enterprise records. Instead, it is used solely to generate a format suitable for analysts by synthesizing the retrieved information.

4.7. Context Construction and Metadata Pruning

After deterministic retrieval, the system produces a task-specific context object that reduces the raw retrieved records to a compact representation containing only the fields required for the selected workflow. This is necessary because enterprise records typically contain substantially more information than is appropriate for a particular analyst's request. The transmission of complete records to the LLM would increase token costs, reveal unnecessary internal metadata, and impair the quality of subsequent output.

Metadata pruning occurs after the appropriate records have been retrieved and integrated. This process eliminates fields that are not required for the current application and retains fields necessary for the bases of factual support, prioritization, or remediation support. For example, the information required for the contextual task of asset triage included the names of affected servers, owners, environment, classification, status with respect to compliance requirements, and degree of risk. Complete inventories of software or full records of prior audit data were not required unless they were directly applicable to the query.

The selected domains vary according to the use case. For UC1, the context object highlights the affected assets and their compliance status. For UC2, it highlights the interplay between the severity of vulnerabilities and asset-specific factors affecting risk. For UC3, the context object emphasizes the feasibility of remediation, including details of patches, owners, deadlines, and indicators for threat hunting. This degree of use-case-specific pruning enabled the retrieval path to reduce the volume of information delivered to the LLM without compromising the data required to generate the final response.

The context object also disentangles factual information from the synthesized reasoning. Factual components are retained as structured key-value pairs, and the LLM is used only to synthesize these components into a summary of explanation, justification, or remediation. This approach facilitates the evaluation of the degree to which the generated response is supported by the retrieved data and enables validation in which the required components are verified prior to the delivery of the response to the analyst.

Table 4.4 illustrates how raw fields are reduced to task-specific context fields.

Table 4.4: Examples of task-specific metadata pruning.

Use case	Retained context	Pruned or deprioritized context
UC1: Contextual asset triage	CVE identifier, affected servers, owners, environment, classification, compliance status, risk level, control findings	Unrelated installed software, unrelated audit dates, unused IoC categories
UC2: Risk-score justification	CVSS score, severity, exploit maturity, asset classification, environment, exposure indicators, compliance score, risk level	Full remediation text, unrelated software inventory, unrelated controls
UC3: Remediation guidance	Patch availability, patch version, affected software, owners, support group, remediation deadline, IoCs, MITRE ATT&CK techniques	Unused compliance-control details, unrelated asset metadata, fields not needed for remediation

A simplified context object can be represented as follows:

```
{
  "task": "risk_justification",
  "vulnerability": {
    "cve_id": "...",
    "severity": "...",
    "cvss_score": "...",
    "exploit_maturity": "..."
  },
  "asset_context": {
    "asset_id": "...",
    "environment": "...",
    "classification": "...",
    "owner": "..."
  },
  "compliance_context": {
    "compliance_status": "...",
    "overall_compliance_score": "...",
    "risk_level": "..."
  }
}
```

This representation is deliberately less extensive than the original records and includes only the information required for the current reasoning task. Noncontributing fields are omitted to facilitate token efficiency, as described in Chapter 3 and to minimize exposure to sensitive internal metadata.

Context construction also primes the response for validation purposes. Because the context object conforms to a known schema, the framework can verify that the required fields are present prior to invoking the LLM or returning the final result. Absent required fields should be reflected as uncertainty and not silently replaced by synthesized information. This approach is critical for prioritizing vulnerabilities, since an incomplete response should be managed in a fundamentally different manner than a confidently generated, unsupported result.

Metadata pruning provides the interface between deterministic retrieval and LLM-based reasoning. It maintains the information needed for explanation and eliminates fields that increase the size, noise, or difficulty of the audit prompt. As a result, the model reasons a compact representation of the enterprise context rather than raw operational data.

4.8. LLM-Based Synthesis Path

The LLM-based synthesis path provides analyst-facing explanations based on the structured context object generated by the orchestration layer. The model does not access factual records, infer relationships among data sources, or determine which enterprise systems should be queried. Instead, it receives selected information about vulnerabilities, assets, ownership, exposure, and compliance. This restricts the model to synthesis and explanation while preserving deterministic control over the factual evidence.

The role of the LLM varies by use case. For contextual asset triage, it summarizes retrieved asset and compliance information in a format suitable for analysts. For risk-score justification, it explains why a prioritization decision follows from the combination of vulnerability severity, exploitability, asset classification, exposure, and compliance status. For remediation guidance, it converts information about patches, affected hosts, ownership, remediation deadlines, and indicators of compromise into an operational summary.

The synthesis path is deliberately limited. This limitation is necessary because LLMs may generate plausible but unsupported statements when asked to extrapolate from incomplete information. To reduce this risk, the framework instructs the model not to infer unavailable fields and to represent missing information as uncertainty. For example, the absence of an ownership field or compliance score in the retrieved context should result in an explicit indication of unavailability rather than an inferred value.

The input to the LLM consists of three components: an analyst task, a structured context object, and a task-specific output schema. The analyst task specifies the expected output, such as a triage summary or remediation report. The context object contains the retrieved evidence, and the output schema defines the required structure of the generated text. This arrangement constrains the model's behavior while retaining its ability to produce coherent explanations.

Table 4.5: Role of the LLM-based synthesis path per use case.

Use case	LLM input	LLM output
UC1: Contextual asset triage	Affected assets, owners, environment, classification, compliance status, and risk level	Analyst-facing triage summary with affected systems and contextual risk indicators
UC2: Risk-score justification	CVSS score, severity, exploit maturity, asset classification, environment, and compliance posture	Natural-language justification for the assigned priority rating
UC3: Remediation guidance	Patch availability, patch version, affected hosts, owners, deadlines, IoCs, and MITRE ATT&CK techniques	Remediation summary and threat-hunting guidance for the analyst

The synthesis path also helps analysts communicate prioritization decisions consistently. Security analysts often need to translate technical vulnerability information into operational explanations for infrastructure owners or remediation teams. Structured explanations can reduce the manual effort required for this communication, but they do not replace analyst judgment. The final responsibility for prioritization decisions remains with the analyst, who can inspect both the generated explanation and the retrieved evidence that supports it.

4.9. Validation and Output Formatting

After generation, the system verifies and formats the responses before returning them to the analyst. This process separates fully developed outputs, incomplete outputs, and unsupported generated content. Verification ensures the completeness of the required fields, adherence to expected formats, and fulfillment of task-specific constraints.

The first phase of validation is schema validation. Each use case specifies a set of expected fields. Contextual triage outputs should include information on affected assets, ownership, compliance status, and risk indicators. Outputs for remediation should include information on affected hosts, ownership of remediation, urgency, details of patches applied, and availability of indicators of compromise. If the required fields are missing from the retrieved context, the system should preserve this uncertainty rather than allowing the LLM to infer the missing values.

Structured outputs may also require format-specific validations. For example, this framework can produce remediation tables and Microsoft Sentinel KQL queries. These results must conform to the expected syntax and structure to be useful to analysts. Importantly, the resulting KQL should invoke only valid fields and retain their inspectability prior to execution.

Table 4.6: Validation checks applied before returning analyst-facing output.

Validation type	Purpose
Required-field check	Verifies that expected fields, such as affected assets, owners, compliance status, and risk level, are present.
Source-grounding check	Confirms that the generated claims are supported by the retrieved context rather than invented by the model.
Output-schema check	Ensures that the response follows the expected structure for the selected use case.
Syntax check	Applies task-specific validation, such as checking generated query structure.
Uncertainty handling	Marks missing or incomplete context explicitly rather than silently replacing it with the generated content.

Output formatting represents the final component of the architecture. The system translates validated results into forms suitable for analyst use. Depending on the use case, this may be a triage table, risk-justification paragraph, remediation checklist, or structured report. The formatting stage is intentionally separated from synthesis so that presentation can be adjusted without changing the retrieval, pruning, or validation logic.

This separation also supports auditing. The generated response can be examined together with the underlying context object and validation results. If a recommendation is disputed, the modular structure makes it possible to determine whether the error originated during retrieval, context integration, synthesis, or output formatting.

4.10. Security and Trust Boundaries

The architecture is intended for use in a regulated financial environment, where security and control requirements directly shape the design. The first trust boundary separates deterministic processing of enterprise data from LLM-based synthesis. Enterprise data are accessed only through controlled retrieval functions, and the LLM does not have unrestricted access to the underlying databases. The model also does not independently select which enterprise systems to query.

The second trust boundary concerns the amount of internal metadata provided to the LLM-based synthesis component. Metadata pruning before model invocation reduces unnecessary exposure of sensitive information and supports the principle of least privilege. This aligns with zero-trust and security-control principles, according to which access to enterprise resources should be restricted, contextual, and controlled rather than implicit [45], [46]. For example, a task may require asset classification and owner group information, but not complete asset records or unrelated software inventories.

The third trust boundary concerns operational action. The framework does not perform remediation actions, apply patches, modify configurations, or invoke automated response procedures. Instead, it provides analyst-facing decision support. This boundary ensures that explanations and remediation recommendations remain subject to human review before any operational action is taken. These controls also reduce LLM-specific risks, such as prompt injection, sensitive information disclosure, and overreliance on generated outputs [50].

Table 4.7: Trust boundaries in the proposed architecture.

Boundary	Controlled risk	Architectural mechanism
Data-source boundary	Unrestricted access to enterprise systems	Deterministic retrieval functions and predefined schemas
Context boundary	Unnecessary exposure of internal metadata	Metadata pruning and task-specific context construction
Synthesis boundary	Unsupported or hallucinated conclusions	Evidence-linked context objects and output validation
Operational boundary	Autonomous remediation without approval	Analyst-facing decision support and human review

These boundaries allow the architecture to use the LLM for explanation and synthesis while maintaining control over factual retrieval and operational decision-making. The resulting design supports the governance restrictions described in Chapter 2 and the non-functional requirements specified in Chapter 3.

5 Implementation

This chapter describes the proof-of-concept implementation used to evaluate the proposed architecture. The implementation realizes the framework described in Chapter 4 in an approved offline environment and compares two orchestration paradigms: the proposed parametric framework and an autonomous-agent baseline. The chapter describes the technical stack, data sources, tool interface, workflow implementation, context construction, validation and logging mechanisms, and reproducibility provisions.

5.1. Implementation Overview

The proof of concept implements two orchestration paradigms over the same benchmark data. The first paradigm is the proposed parametric framework, which uses fixed workflow templates and restricted tool access. The second paradigm is an autonomous-agent baseline, which receives the same data and tools but determines the sequence of tool calls during execution. Both paradigms operate on the same approved static production-derived CSV snapshot and are evaluated using the same benchmark tasks.

The implementation follows the architectural separation introduced in Chapter 4. Factual retrieval is performed by deterministic tool functions. Retrieved records are then aggregated, pruned, and converted into task-specific context objects before LLM-based synthesis is applied. This separation allows the benchmark to compare the effect of orchestration strategy on accuracy, efficiency, and robustness.

The proof of concept supports three operational use cases: contextual asset triage, risk-score justification, and remediation guidance with threat-hunting query support. These use cases correspond to the benchmark design described in Chapter 3 and require different combinations of retrieval, context construction, synthesis, validation, and output formatting.

5.2. Technical Stack

The proof-of-concept system was implemented as an offline Python application. This setup allowed the benchmark to be executed within the approved internal environment without relying on external public APIs or live production databases. The purpose of this implementation was not to optimize database performance, but to evaluate the orchestration behavior of the proposed framework under controlled and reproducible conditions.

LLM-based synthesis was performed using an internally approved deployment of Claude Opus 4.6. The model was used only after task-specific context had been retrieved, pruned, and assembled by the orchestration layer. It did not access the underlying CSV files, database schemas, or enterprise systems directly.

The implementation used approved static production-derived CSV snapshot sources. Each retrieval function loaded and queried these files through predefined schemas. This design preserved the relationships required for evaluation while avoiding direct interaction with live vulnerability-management, CMDB, or compliance systems.

Randomness in the primary benchmark run was controlled using a fixed seed. The robustness sensitivity analysis repeated the experiment across multiple random seeds, allowing the evaluation to distinguish the effect of orchestration design from variation caused by injected stochastic workflow noise.

5.2.1. Model Configuration

The LLM was configured as a bounded synthesis component. It did not directly access data files, database schemas, or enterprise systems. All factual information was retrieved through the tool interface, converted into structured context objects, and then passed to the model for analyst-facing synthesis.

The proof-of-concept implementation used a bank-approved offline deployment of Claude Opus 4.6. This deployment was selected because it satisfied the internal security and compliance conditions for the evaluation environment. The results should therefore be interpreted in the context of a regulated financial institution, where model adoption depends on security approval, deployment constraints, and administrative controls.

For each use case, the model generated context-based summaries, risk justifications, or remediation-oriented guidance. It was instructed to use only retrieved context and to mark unavailable information explicitly rather than infer missing owners, exposure status, compliance scores, or remediation dates.

This configuration preserved the separation between deterministic retrieval and probabilistic synthesis. Factual data selection, schema joins, context reduction, validation, and logging remained under programmatic control, while the LLM was used to produce structured natural-language explanations for analysts.

5.3. Data Sources and Snapshot Schemas

Three approved static production-derived CSV snapshot sources provide the foundation for this proof of concept: a vulnerability dataset, a ServiceNow-style CMDB export, and compliance records from the bank's internal Network Security Framework (NSF). These CSV files were exported from the selected live operational systems and subsequently "frozen" for offline analysis. The benchmark is therefore based on production-derived records for the selected use cases and eliminates all direct access to production databases, scanners, CMDB systems, or platforms for compliance reporting. Field values that could disclose sensitive infrastructure, personnel, IP addresses, or operational details were omitted, generalized or partially obfuscated.

The vulnerability database contains CVE records enriched with severity, CVSS scores, affected software, patch information, exploit maturity, indicators of compromise, and MITRE ATT&CK techniques. These fields are used for prioritization, risk justification, remediation guidance, and threat-hunting query generation purposes.

The CMDB contains server records with identifiers, hostnames, operating systems, ownership metadata, support groups, environment labels, asset classifications, installed software, and associated CVEs. The field `cves_affecting` enables the framework to join the vulnerability records to the affected assets.

The NSF compliance dataset contains per-asset compliance records. Each record includes an asset identifier, compliance score, compliance status, control-level findings, risk level, and a remediation deadline. The field `asset_id` joins compliance records with CMDB records through the CMDB `sys_id` field.

Table 5.1: Approved static production-derived CSV snapshot sources used by the proof-of-concept implementation.

Data source	Records	Purpose
CVE database	5	Provides vulnerability descriptions, CVSS scores, patch information, exploit maturity, IoCs, and MITRE ATT&CK techniques.
ServiceNow-style CMDB	6	Provides asset metadata, ownership, environment, classification, installed software, and CVE-to-asset relationships.
NSF compliance records	6	Provides compliance status, control findings, risk level, and remediation deadlines for affected assets.

5.4. Tool Interface

Data access is implemented through a limited set of tool functions. Each function performs a controlled operation on the approved static production-derived CSV snapshot. The LLM has no direct access to the underlying snapshot files. Both orchestration paradigms use the same tool interface, so differences in performance can be attributed to orchestration behavior rather than unequal access to data.

This approach supports the security and administrative requirements of the deployment environment. The assessment of retrieval and inference is based on the approved static production-derived CSV snapshot and not on direct access to the live databases. Consequently, changes to the design of the orchestration layer can be tested without requiring additional modifications to production implementations or the initiation of new cycles for the approval of access to live databases.

The primary retrieval tools include CVE lookup, CMDB queries, NSF compliance queries, risk analysis, KQL generation, and results validation. These represent the operations required to support the three benchmark use cases. The parametric execution of these operations is achieved in fixed sequences by the parametric framework and with greater flexibility by the autonomous baseline.

Table 5.2: Main tool functions exposed by the proof-of-concept implementation.

Tool function	Purpose
`lookup_cve`	Retrieves vulnerability information for a given CVE identifier.
`query_cmdb`	Retrieves server records affected by a given CVE or matching asset metadata.
`query_nsf`	Retrieves compliance and control information for a given asset identifier.
`perform_delta_analysis`	Combines vulnerability severity with asset and compliance context to support priority justification.
`generate_sentinel_kql`	Generates Microsoft Sentinel KQL using indicators of compromise from the vulnerability record.
`validate_query_syntax`	Checks whether generated query output satisfies expected syntax and structure constraints.

5.5. Parametric Orchestration Implementation

The parametric orchestration implementation applies predefined workflow templates to each supported use case. The term *parametric* indicates that retrieval is controlled by explicit task parameters and programmatic logic rather than by allowing the LLM to select its own retrieval strategy.

Each workflow template specifies the required tool calls, their execution order, the expected input and output fields, and the context object to be created before LLM-based synthesis. This ensures that factual retrieval is completed before an analyst-facing explanation is generated.

For contextual asset triage, the workflow uses multi-hop retrieval. It first retrieves the relevant CVE record, then queries the CMDB to identify affected assets, and finally retrieves NSF compliance records for each affected asset. This sequence is necessary because omitting the compliance step would remove the evidence needed to assess compliance posture and control risk, even if the response still identified the affected servers.

For risk-score justification, the workflow follows a more compact retrieval path. It combines vulnerability severity, CVSS score, exploit maturity, asset classification, environment, and compliance risk. These fields are then used to support a deterministic delta-analysis or risk-mapping step before the LLM generates an analyst-readable justification.

For remediation guidance, the workflow retrieves patch availability, patch version, affected software, affected assets, asset owners, remediation deadlines, indicators of compromise, and MITRE ATT&CK techniques. The resulting context object is used to generate an analyst-facing remediation summary and, where applicable, threat-hunting query support.

Table 5.3 summarizes the implemented workflows.

Table 5.3: Parametric workflow templates implemented in the proof-of-concept.

Use case	Required retrieval steps	Output produced
UC1: Contextual asset triage	`lookup_cve` → `query_cmdb` → `query_nsf` per affected asset	Triage summary containing affected assets, owners, compliance status, and risk context
UC2: Risk-score justification	Retrieve vulnerability attributes, asset classification, environment, and compliance risk indicators	Priority justification explaining the difference between generic severity and organization-specific risk
UC3: Remediation guidance	Retrieve patch information, affected hosts, owners, deadlines, IoCs, and MITRE ATT&CK techniques	Remediation summary and threat-hunting query support

The implementation of the parametric workflow is summarized as follows:

```
def run_parametric_workflow(question):
    task = classify_task(question)

    if task == "asset_triage":
        cve = lookup_cve(question.cve_id)
        assets = query_cmdb(cve.cve_id)
        compliance = [query_nsf(asset.sys_id) for asset in assets]
        context = build_triage_context(cve, assets, compliance)
        return synthesize_triage_report(context)

    if task == "risk_justification":
        cve = lookup_cve(question.cve_id)
        asset = query_cmdb(question.asset_id)
        compliance = query_nsf(asset.sys_id)
        context = build_risk_context(cve, asset, compliance)
        return synthesize_risk_justification(context)

    if task == "remediation_guidance":
        cve = lookup_cve(question.cve_id)
        assets = query_cmdb(cve.cve_id)
        compliance = [query_nsf(asset.sys_id) for asset in assets]
        context = build_remediation_context(cve, assets, compliance)
        return synthesize_remediation_report(context)
```

The main advantage of this implementation is that the retrieval sequence is not planned by the LLM. Instead, the model receives the output of a completed retrieval workflow after the relevant records have been retrieved and compressed into a structured context object. This supports deterministic processing, auditability, and control over data boundaries.

The parametric approach also limits the tools available for each use case. For example, a triage workflow does not require schema-discovery tools once the required schema relationships have been encoded in the workflow. Restricting the available tools reduces unnecessary exploration and makes the execution trace easier to inspect.

Finally, the parametric implementation creates a structured intermediate state before generating the final response. This state contains the retrieved CVE fields, affected asset records, compliance information, and task-derived context. Separating this intermediate state from the output text makes it possible to evaluate whether the final answer is supported by the retrieved evidence.

5.6. Autonomous Agent Baseline

The autonomous agent baseline represents a less restrictive alternative for comparison with the parametric approach. In contrast to the parametric framework, an autonomous agent is not provided with a fixed workflow template for each application. Rather, it receives the analyst's question, descriptions of available tools, and instructions for achieving the desired result. Based on these inputs, the model determines which tools to invoke, in what sequence, and when sufficient information has been obtained to provide a response.

The purpose of this baseline is not to represent a deliberately weak system. Autonomous agents can be useful for exploratory tasks in which relevant sources, retrieval order, or required fields are not known in advance. However, the vulnerability-management tasks evaluated in this thesis are comparatively structured: they require specific fields, predictable joins, and well-defined output formats. The autonomous baseline therefore tests whether additional flexibility improves or impairs performance in this type of structured security workflow.

The autonomous agent has greater access to tools than the parametric approach does. In addition to its primary capabilities for retrieval and synthesis, it may also use auxiliary tools such as schema enumeration or syntax validation. This exemplifies a typical design strategy for agents in which the model can examine available tools and develop its own plan of action. The result is that the model may generate excessive amounts of exploratory calls or omit the necessary retrieval steps required for a complete response.

The baseline follows a generic loop:

```
def run_autonomous_agent(question):
    state = initialize_agent_state(question)

    while not state.finished:
        tool_call = llm_select_next_tool(state)
        tool_result = execute_tool(tool_call)
        state = update_agent_state(state, tool_call, tool_result)

    return llm_generate_final_answer(state)
```

In this loop, the large language model regulates the retrieval strategy. The model can examine schemas, validate an intermediate query, retrieve a CVE, identify affected assets, obtain compliance data, or generate a final answer. All tool calls and outputs were logged to allow a comparison of the fully autonomous execution with the parametric execution.

The autonomous baseline differs from the proposed approach in three major ways. First, the workflow order was determined by the model rather than by prior specifications. Second, the agent has greater access to tools that promote exploratory behavior and increase the number of possible execution paths. Third, the final output may originate from an intermediate state, the degree of completeness of which reflects the model's own planning decisions.

Table 5.4 summarizes the differences in implementation between the two paradigms.

Table 5.4: Implementation differences between parametric orchestration and the autonomous baseline.

Dimension	Parametric orchestration	Autonomous baseline
Workflow control	Defined by use-case-specific templates	Planned by the LLM during execution
Tool access	Restricted to tools required by the selected workflow	Broader tool access, including exploratory or auxiliary tools
Retrieval order	Programmatic and fixed for each use case	Determined dynamically by the model
Context construction	Built from predefined schemas after deterministic retrieval	Emerges from the sequence of model-selected tool calls
LLM role	Synthesis and explanation after retrieval	Planning, retrieval strategy, and final synthesis
Main implementation risk	Limited flexibility outside predefined workflows	Skipped hops, redundant calls, incomplete context, or unsupported synthesis

This comparison evaluates the trade-off between flexibility and control. A generic autonomous agent can accommodate a wider range of possible questions and requires fewer predefined workflow templates. However, this flexibility also makes execution paths less predictable. In regulated financial-security workflows, where vulnerability-prioritization outputs must be traceable and supported by evidence, this reduced predictability is an important implementation risk.

Both paradigms were applied to the same approved static production-derived CSV snapshot and evaluated using identical benchmark tasks. The comparison therefore isolates orchestration strategy as the main implementation difference. The two paradigms differ in how they control retrieval, construct intermediate context, and allocate autonomy to the LLM during execution.

5.7. Context Construction and Prompt Assembly

After completing the information retrieval steps, the implementation generates a task-specific context object that disentangles factual information from synthesized explanations. The retrieved records for CVE, CMDB, and NSF were not provided to the language model as raw rows from a database. Instead, the implementation extracts the fields required for the current use case and presents them in a compact format.

The context-construction phase is common to both orchestration paradigms but is implemented differently. In the parametric approach, the context object is constructed after completing a pre-specified retrieval workflow. In contrast, for the autonomous baseline, the extent of the available context for the model is determined by the sequence of tool calls chosen during execution. This resulted in an important difference in the completeness of the final prompt, which reflects the extent to which all required retrieval steps were achieved.

For contextual asset triage, the context object includes the vulnerability identifier, affected assets, asset owners, environments, classifications, compliance status, compliance risk, and associated control results. To justify the risk scores, the context object reflects the interdependence of vulnerability severity and asset-specific risk factors. To support remediation decisions, the context object provides details on patch availability, affected software, owners, deadlines for remediation, indicators of compromise, and MITRE ATT&CK techniques.

Table 5.5: Context fields used during prompt assembly.

Use case	Main context fields	Purpose
UC1: Contextual asset triage	CVE identifier, affected servers, owners, environment, classification, compliance status, risk level, control findings	Provide an analyst-facing overview of affected assets and their operational risk context
UC2: Risk-score justification	CVSS score, severity, exploit maturity, asset classification, environment, compliance score, and risk level	Explain why the organization-specific priority differs from or confirms the generic severity
UC3: Remediation guidance	Patch availability, patch version, affected software, owners, support group, remediation deadline, IoCs, and MITRE ATT&CK techniques	Produce actionable remediation guidance and threat-hunting support

Prompt assembly integrates three elements: task instruction, structured context object, and desired output format. The task instructions described the type of response to be generated. The context object incorporates the retrieved evidence. The output format defines the response structure, such as a triage summary, risk justification, remediation checklist, or KQL-oriented response.

The simplified prompt structure is as follows:

```
Task:
Generate a risk justification for the given vulnerability and assets.

Retrieved context:
{
  "vulnerability": {
    "cve_id": "...",
    "severity": "...",
    "cvss_score": "...",
    "exploit_maturity": "..."
  },
  "asset": {
    "asset_id": "...",
    "environment": "...",
    "classification": "...",
    "owner": "..."
  },
  "compliance": {
    "compliance_status": "...",
    "overall_compliance_score": "...",
    "risk_level": "..."
  }
}

Output requirements:
- Explain the priority of the decision.
- Only the retrieved context was used.
- State explicitly if the required information is missing.
- Do not invent owners, compliance scores, or exposure status.
```

Explicit prompt instructions reduce the risk of unsupported inference. When a required field is missing from the context object, the response should state that the information is unavailable rather than provide a plausible but unsupported value. This is especially important in vulnerability management, where an explicit uncertainty is safer than a confident but incorrect claim about ownership, exposure, compliance, or remediation status.

Prompt assembly therefore forms part of the control surface of the framework. The prompt does more than request an answer: it defines the evidence available to the model, constrains how missing information should be handled, and specifies the expected output structure.

5.8. Validation and Logging

The implementation includes validation and logging mechanisms to support inspection of the execution trace. Validation checks whether the output satisfies the structural requirements of the selected use case and whether generated claims remain grounded in retrieved context. Logging records tool calls, retrieved records, constructed context objects, token metrics, injected noise events, and final outputs. Together, these mechanisms support reproducibility, auditing, and error analysis.

Validation was performed at two points within the workflow. Prior to invoking the LLM, the implementation verifies that all required contextual fields have been obtained. For example, a contextual triage response includes information about affected assets, owners, compliance status, and risk level. The absence of these fields results in either the presentation of the missing information as uncertainty or the prevention of confident generation of an explanation by the framework.

The second validation step occurs after the LLM generates a response. Here, the implementation verifies that the response includes all expected components and that the derived assertions are consistent with the retrieved information. Additional validation is applied to structured outputs, such as drafts of Microsoft Sentinel KQL queries. These procedures do not establish the semantic completeness of the resulting query but identify missing elements and obvious errors of structure prior to the delivery of the output to the analyst.

Table 5.6: Validation checks implemented in the proof-of-concept.

Validation check	Description
Required-field validation	Checks whether the context object contains the fields required by the selected use case.
Grounding validation	Checks whether generated claims correspond to the retrieved fields rather than unsupported model outputs.
Output-structure validation	Checks whether the generated response follows the expected structure for triage, justification, or remediation.
Query-structure validation	Checks whether the generated KQL-oriented output contains the expected query structure and referenced indicators.
Missing-context handling	Ensures that the unavailable context is represented as missing rather than being silently filled by the generated content.

Logging was used to enable reproducibility and subsequent analysis. For each benchmark question, the implementation logs the selected orchestration paradigm, invoked tool calls and associated arguments, retrieved records, token counts, and final answers. The experiments with robustness also log injected events of noise so that errors in the final output can be attributed to missed hops, lost records, incorrect parameters, hallucinations, additional tool calls, or missing fields.

Table 5.7: Logged information for each benchmark run.

Logged item	Purpose
Benchmark question identifier	Links each output to the corresponding use case and expected answers.
Orchestration paradigm	Identifies whether the run used parametric orchestration or the autonomous baseline.
Tool-call sequence	Records which tools were called and in which order.
Tool arguments and outputs	Enables inspection of retrieval correctness and intermediate state.
Constructed context object	Shows what evidence was made available to the LLM.
Generated answer	Stores the final analyst-facing output.
Token and latency metrics	Supports operational-efficiency analysis.
Noise events	Supports robustness analysis by linking injected errors to output degradation.

The logging mechanism is not intended to duplicate the production audit environment. Rather, it provides sufficient trace information to permit the analysis of the behavior of this proof of concept. This achieved the experimental objective of comparing orchestration paradigms in a controlled environment.

5.9. Reproducibility

Because the approved static production-derived CSV snapshots contain confidential information about bank operations, they cannot be included in the public artifacts associated with this thesis. Reproducibility is therefore supported at two levels. First, the benchmark questions, scoring criteria, tool interfaces, noise conditions, and aggregated results are documented. Second, the accompanying package provides anonymized, structurally faithful examples and benchmark outputs that replicate the experimental design without revealing operational records. The reported results are based on the approved snapshots, whereas public or shared example records should be interpreted as illustrative substitutes.

The proof-of-concept was intended to be reproducible in an approved offline testing environment. Benchmark execution is based on an approved static production-derived CSV snapshot and does not require interaction with external public APIs. The large language model component was implemented with a bank-approved offline deployment of Claude Opus 4.6, which was minimally adapted for evaluation purposes. This enables the assessment of orchestration behavior while maintaining control over the boundaries of data flow and eliminating reliance on active connections to live production databases.

The robustness assessment employed seeded randomness. For each benchmark execution, a random seed was fixed to generate an identical sequence of stochastic noise events. This is necessary because the noise model incorporates the probabilistic occurrences of missed hops for tool calls, lost records at the server, incorrect parameters, fabricated lines, additional tool calls, and missing data fields.

The benchmark can be executed using the following command:

```
python main.py benchmark --offline --noise --verbose
```

The implementation produces three main output artifacts:

- `results/benchmark_results.json`: full per-question benchmark data;
- `results/benchmark_results.csv`: tabular summary of benchmark results;
- `results/benchmark_charts.html`: generated benchmark charts.

These artifacts allow the evaluation in Chapter 6 to be inspected and replicated. The JSON output captures all detailed information about the run, the CSV output supports tabular analysis, and the graphical output provides a visual inspection of the benchmark results.

The implementation has two limitations in terms of reproducibility. First, a fixed offline model deployment and fixed prompt templates were used to increase the control and achieve greater reproducibility.

However, the results depend on the particular version of the model, its configuration, and the deployment conditions. Second, the framework provides access to approved static production-derived CSV snapshots rather than real-time connections to production databases. The snapshots are representative of the selected applications at the time of export but do not reflect subsequent changes in asset status, ownership, risk exposure, or degree of compliance. The limitations are discussed in greater detail in Chapter 7.

To ensure reproducibility, the implementation should record the model identifier, model configuration, prompt-template version, random seed, stochastic-noise profile, benchmark-question identifier, scoring-rubric version, execution time, and implementation version for each benchmark run. Recording this information makes it easier to distinguish variation caused by orchestration design from variation caused by model configuration, prompt changes, or benchmark execution conditions.

6 Evaluation

This chapter evaluates whether the proposed parametric orchestration framework improves the accuracy, efficiency, robustness, and estimated operational cost of LLM-supported vulnerability prioritization compared with an autonomous-agent baseline. The evaluation is organized around five experimental questions:

EQ1: Does the parametric orchestration framework improve overall accuracy compared with the autonomous-agent baseline?

EQ2: Which vulnerability-management use cases benefit most from constrained orchestration?

EQ3: Does constrained orchestration reduce operational overhead in terms of tool calls and token consumption?

EQ4: How robust are the orchestration paradigms under stochastic workflow noise?

EQ5: What are the expected operational cost implications of the two orchestration paradigms?

The following sections answer these questions sequentially. Section 6.2 addresses EQ1, Section 6.3 addresses EQ2, Section 6.4 addresses EQ3, Section 6.5 addresses EQ4, and Section 6.6 addresses EQ5. Section 6.7 provides a qualitative error analysis to explain the observed differences.

6.1. Evaluation Overview

The comparison evaluates the two orchestration paradigms under identical benchmark conditions. The first paradigm is the proposed parametric framework with fixed workflow templates, limited access to tools, and deterministic policies for data retrieval. The second paradigm represents an autonomous agent baseline with greater flexibility in tool access and complete control over the order of tool calls and data retrieval strategies during the execution.

Both paradigms operate on the same approved static production-derived CSV snapshot and address identical benchmark queries. The snapshots reflect the actual operational data for the selected use cases but are not a live interface to production databases. The use of identifying information is limited to protect security and maintain confidentiality. Both orchestration strategies employed identical bank-approved offline implementations of Claude Opus 4.6 and identical tool interfaces. The model version represents the approved internal implementation available at the time of evaluation, not the most recent publicly available model. This avoided variations in model capacity or data availability and permitted the evaluation of the effect of the orchestration strategy.

The benchmark encompasses three domains of data: records of vulnerabilities, ServiceNow-style records of configuration management database assets, and records of compliance with NSF requirements. For each question, the system must integrate, synthesize, or otherwise derive information from these data sources.

The results are reported in terms of the weighted accuracy score defined in Equation 3.1. This score reflects the correct identification of affected servers, correct attribution of ownership to the affected assets, and appropriate representation of the expected supporting fields. In addition, the report measures operational efficiency, including the number of tool invocations and tokens consumed. The system robustness was assessed using the stochastic noise model described in Section 3.3.1.

The evaluation was organized as follows. Section 6.2 reports the overall benchmark accuracies of both orchestration paradigms. Section 6.3 analyzes the performance across the three use cases. The subsequent sections compare the efficiency, robustness, financial impact, and qualitative failure modes of the two orchestration paradigms.

6.2. Overall Accuracy

Table 6.1 shows the overall benchmark accuracy for the two orchestration paradigms. The parametric framework yielded a mean accuracy of 98.4%, whereas the autonomous baseline achieved a mean accuracy of 90.0%. This corresponds to an 8.4 percentage-point difference in favor of the parametric framework. The per-question ground truth and score components used to compute these aggregate values are reported in Appendix A.

Table 6.1: Overall benchmark accuracy of the two orchestration paradigms.

Approach	Mean accuracy	Observation
Parametric orchestration	98.4%	Only one benchmark question scored below 100%
Autonomous baseline	90.0%	Five benchmark questions scored below 100%

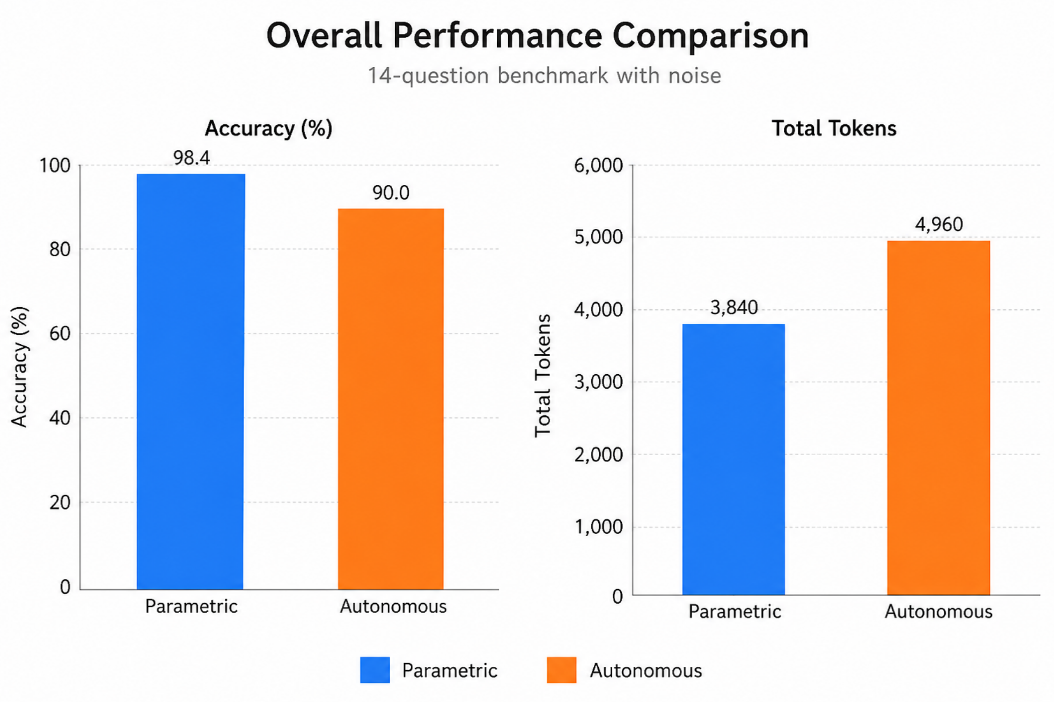

Figure 6.1: Overall performance comparison across the 14-question benchmark. The parametric framework achieved higher mean accuracy while consuming fewer total tokens than the autonomous baseline.

The overall results in Table 6.1 and Figure 6.1 show that the parametric framework achieved higher mean accuracy while also consuming fewer total tokens than the autonomous baseline. This result supports the central architectural argument of this thesis: for structured vulnerability-management tasks, reliability depends less on giving the model freedom to explore and more on ensuring that the correct evidence is retrieved before synthesis begins.

This difference is not attributable to unequal access to information, since both paradigms used the same benchmark data and tool interface. Instead, the performance gap reflects how each paradigm constructs the evidence base for the final answer. The parametric framework follows predefined retrieval paths, which makes the required joins between vulnerability records, affected assets, ownership metadata, and compliance status more consistent. The autonomous baseline can choose its own sequence of tool calls, but Figure 6.1 suggests that this flexibility did not translate into higher accuracy for the selected benchmark. In this setting, autonomous exploration introduced more opportunity for evidence to be retrieved incompletely or integrated inconsistently.

The overall comparison therefore supports the decision to separate deterministic retrieval from LLM-based synthesis. The observed advantage should not be interpreted as evidence that the language model became intrinsically better at reasoning about vulnerabilities. Rather, the parametric framework improved the conditions under which synthesis occurred by providing a more complete and controlled evidence base before the response was generated.

6.3. Use-Case-Level Results

Table 6.2 and Figure 6.2 show the accuracy breakdown by use case. The largest differences in use-case accuracy were observed for UC1 (contextual asset triage), where the parametric framework achieved 96.2% accuracy compared with 77.7% for the autonomous baseline. Both approaches achieved perfect accuracy for UC2 (risk-score justification). In UC3 (remediation guidance and KQL generation), the parametric framework achieved 100.0%, while the autonomous baseline achieved 98.5%.

Table 6.2: Accuracy by benchmark use case.

Use case	Parametric	Autonomous	Difference
UC1: Contextual asset triage	96.2%	77.7%	+18.5 pp
UC2: Risk-score justification	100.0%	100.0%	0.0 pp
UC3: Remediation guidance and KQL generation	100.0%	98.5%	+1.5 pp

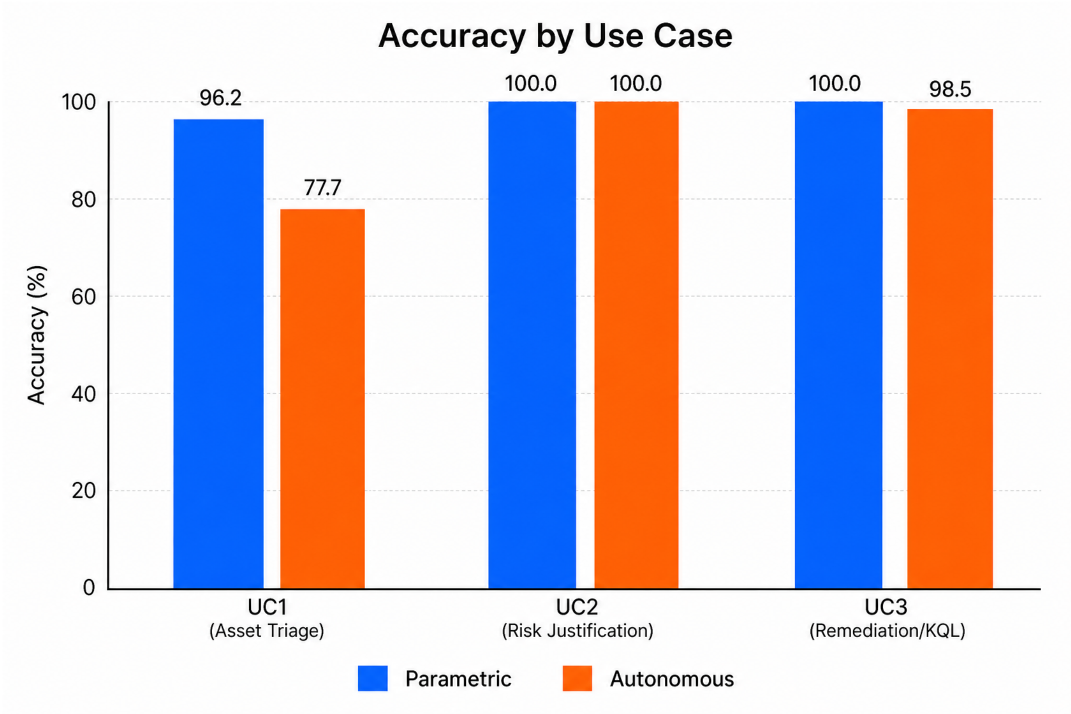

Figure 6.2: Accuracy by benchmark use case. The largest difference between the two orchestration paradigms appears in UC1, where contextual asset triage requires multi-hop retrieval across vulnerability, CMDB, ownership, and compliance records.

The UC1 result provides the clearest evidence for the benefit of deterministic orchestration. Contextual asset triage requires a multi-hop evidence chain: the system must start from a CVE, identify affected CMDB assets, and then retrieve ownership and compliance context for those assets. The parametric workflow encodes this chain explicitly, which makes it less likely that a required hop is omitted. The lower autonomous-baseline score in Figure 6.2 indicates that self-directed retrieval was less reliable for this type of dependent retrieval task.

UC2 shows a different pattern. Risk-score justification required a shorter and more structured retrieval path, because the answer mainly depended on combining vulnerability severity with asset-specific and compliance-related context. Both paradigms achieved perfect accuracy for this use case. This suggests that autonomous orchestration can perform well when the relevant context is compact, the required fields are clear, and the output format is highly constrained.

UC3 produced only a small difference between the two paradigms. Remediation guidance and KQL generation required information about patches, affected hosts, owners, remediation context, and indicators of compromise. The autonomous baseline lost a small amount of field information, but its overall performance remained close to that of the parametric framework. This result suggests that the advantage of constrained orchestration is smaller when the task structure is clear and the required evidence chain is less fragile than in contextual asset triage.

Overall, the use-case results show that the advantage of parametric orchestration increases with retrieval complexity. When a task depends on multiple linked records, predefined workflows reduce the risk of incomplete context construction. When the task is narrower and the required evidence is easier to retrieve, the autonomous baseline can achieve comparable accuracy, although later efficiency results show that it often does so with higher overhead.

6.4. Efficiency Analysis

Accuracy alone is insufficient for assessing the suitability of an orchestration framework for enterprise use. A system that achieves high accuracy through excessive tool calls, long prompts, or unnecessary intermediate steps may be difficult to scale in operational settings. The evaluation therefore also compares the operational efficiency of the parametric framework and the autonomous baseline.

Table 6.3 reports the average number of tool calls and tokens per question for each use case. The autonomous baseline consumed more tokens across all three use cases. The clearest efficiency difference appears in UC2: both paradigms achieved identical accuracy, but the autonomous baseline required four times as many tool calls and twice as many tokens per question.

Table 6.3: Efficiency metrics by use case. The autonomous baseline generally uses more tokens and more tool calls, especially in UC2 where it reaches the same accuracy as the parametric framework but with substantially higher overhead.

Use case	Metric	Parametric	Autonomous
UC1	Tool calls per question	8.2	7.5
UC1	Tokens per question	320	347
UC2	Tool calls per question	1.0	4.0
UC2	Tokens per question	160	320
UC3	Tool calls per question	4.0	5.2
UC3	Tokens per question	320	400

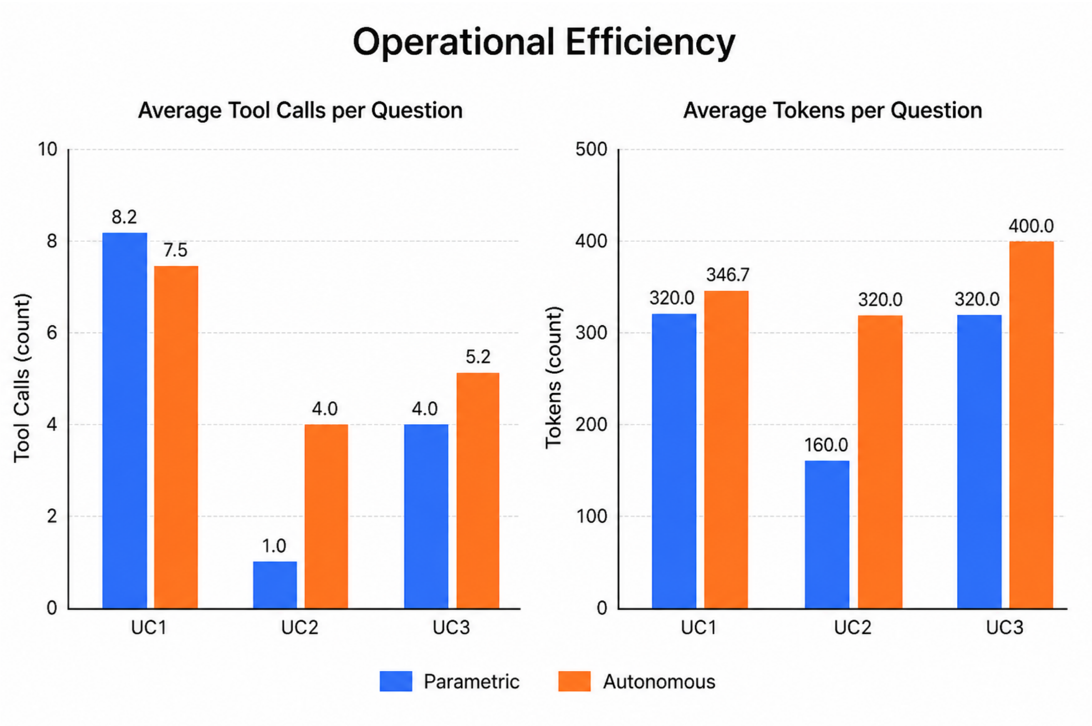

Figure 6.3: Operational-efficiency comparison by use case. The autonomous baseline generally consumes more tokens and performs more exploratory tool use, while the parametric framework follows fixed retrieval paths.

The results in Table 6.3 and Figure 6.3 must be interpreted with caution. This framework makes slightly more tool calls than the autonomous baseline for UC1 because it explicitly attempts all the necessary retrieval steps for each affected asset. Therefore, additional tool calls reflect complete multi-step retrieval and are not indicative of inefficient behavior. In contrast, the fewer tool calls required by the autonomous baseline for UC1 reflect occasional omissions of necessary retrieval steps and ultimately result in lower overall accuracy.

The UC2 results illustrated a different pattern. Risk-based justification represents a lower level of complexity because the required information is retrieved by a compact path and subjected to systematic analysis. Both methods yielded perfect performance for this application; however, the autonomous baseline required additional exploratory interactions to arrive at identical results. Thus, autonomous planning resulted in additional overhead, even for tasks that did not require exploration.

The parametric approach also remains more efficient for remediation workflow. Remediation requires information about patches, affected systems, ownership context, and evidence of compromise. The parametric workflow obtains these data with a fixed sequence of operations, whereas the autonomous baseline may invoke additional tools and generate a more extensive intermediate context. Therefore, the autonomous baseline consumed more tokens and achieved a somewhat lower accuracy.

Table 6.4 summarizes the total token usage across the full benchmark. The parametric framework consumed 3,840 tokens, whereas the autonomous baseline consumed 4,960 tokens. This corresponds to a 29.2% increase in token usage for the autonomous baseline.

Table 6.4: Total token consumption across the 14-question benchmark. The autonomous baseline consumes 29.2% more tokens than the parametric framework.

Metric	Parametric	Autonomous	Difference
Prompt tokens	2,400	3,100	+29.2%
Completion tokens	1,440	1,860	+29.2%
Total tokens	3,840	4,960	+29.2%
Tokens per question	274	354	+80

The efficiency analysis indicates that the parametric framework achieved higher overall accuracy while consuming fewer language-model tokens. This supports the architectural choice to use deterministic retrieval and fixed workflows for structured vulnerability-management tasks. Autonomous orchestration remains attractive for exploratory settings, but in the evaluated workflows it introduced additional token and tool-call overhead without improving accuracy.

6.5. Robustness Under Stochastic Noise

The robustness experiment evaluates how each orchestration paradigm responds to stochastic workflow errors. Using the noise model described in Section 3.3.1, the benchmark injects perturbations into retrieval, tool use, and output generation. The purpose is not to estimate production failure rates, but to assess whether each paradigm remains reliable when workflow execution is imperfect.

Both methods achieved perfect accuracy on the controlled snapshot benchmark without injected noise. The noisy benchmark therefore measures degradation from this ideal baseline. Table 6.5 and Figure 6.4 show that the parametric framework decreased from 100.0% to 98.4%, corresponding to a 1.6 percentage-point degradation. The autonomous baseline decreased from 100.0% to 90.0%, corresponding to a 10.0 percentage-point degradation.

Table 6.5: Accuracy degradation under stochastic noise. The parametric framework degrades less than the autonomous baseline because its fixed workflows reduce the likelihood that required retrieval steps are skipped.

Approach	Noise-free accuracy	Noisy accuracy	Degradation
Parametric orchestration	100.0%	98.4%	-1.6 pp
Autonomous baseline	100.0%	90.0%	-10.0 pp

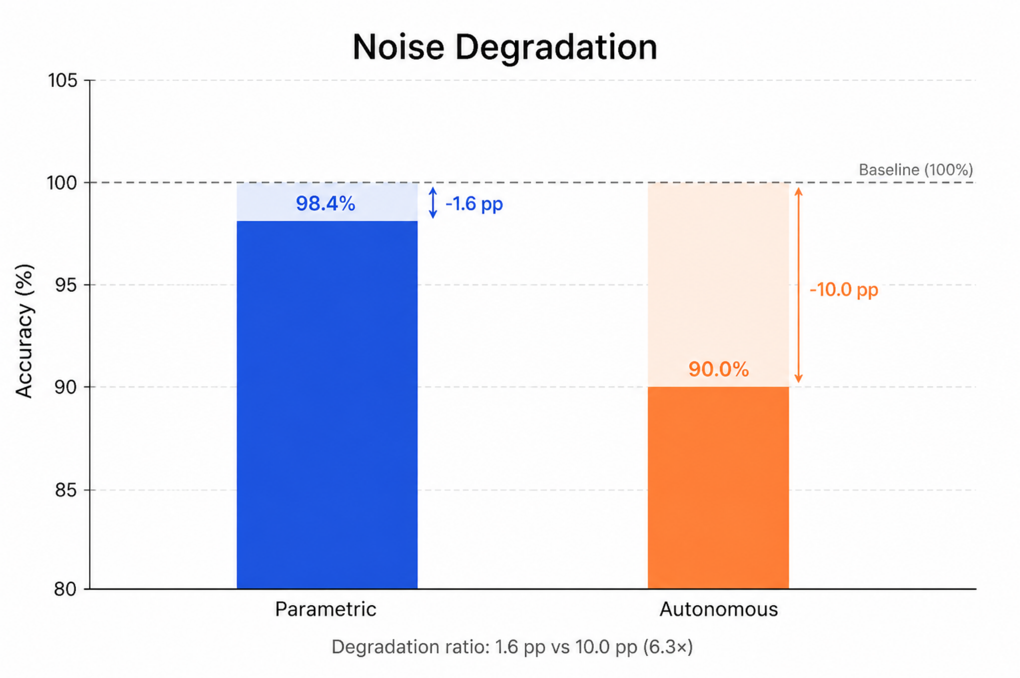

Figure 6.4: Accuracy degradation under stochastic noise. Both orchestration paradigms achieve perfect accuracy without injected noise, but the autonomous baseline exhibits a larger degradation under missed hops, dropped records, wrong parameters, hallucinated lines, extra tool calls, and missed fields.

To assess whether the robustness result depends on a single noise calibration or random seed, the stochastic noise experiment was repeated across three global noise scales and five random seeds, as defined in Section 3.3.2. Table 6.6 reports the mean weighted accuracy and standard deviation for each orchestration paradigm under low-, base-, and high-noise settings.

Table 6.6: Sensitivity analysis over noise scales and random seeds.

Noise scale	Interpretation	Parametric mean $\pm$ std.	Autonomous mean $\pm$ std.	Difference
0.5$\times$	Low-noise setting	97.6% $\pm$ 7.9 pp	92.7% $\pm$ 19.4 pp	+4.9 pp
1.0$\times$	Base setting	94.7% $\pm$ 12.7 pp	85.6% $\pm$ 25.9 pp	+9.1 pp
1.5$\times$	High-noise setting	93.5% $\pm$ 14.8 pp	80.9% $\pm$ 27.7 pp	+12.6 pp

The 1.0$\times$ row in Table 6.6 reports the mean across five random seeds and therefore does not exactly match the single primary base noise run reported in Table 6.5. Table 6.5 presents the main benchmark comparison, whereas Table 6.6 evaluates the robustness across alternative random realizations of the same noise profile.

The sensitivity analysis indicates that the robustness result is not an artifact of a single random seed or a particular noise calibration. Across the low-, base-, and high-noise settings, the parametric framework retained higher mean weighted accuracy and lower variance than the autonomous baseline. The performance gap increased from 4.9 percentage points in the low-noise setting to 12.6 percentage points in the high-noise setting, suggesting that fixed workflow control degrades more gracefully as stochastic workflow errors increase.

The main mechanism is the reduction of cascading retrieval failures. In the autonomous baseline, a missed early retrieval step can remove evidence needed for later parts of the answer. In the parametric framework, the retrieval order is predefined, so later steps are still attempted even when an individual field or record is affected by noise. The framework does not eliminate errors, but it reduces the likelihood that one failure prevents the construction of the entire evidence chain.

This effect is most visible in UC1, where contextual asset triage depends on multiple linked retrieval steps. The system must identify affected assets, attach ownership and compliance context, and preserve the expected supporting fields. Errors in this chain have a larger effect on the final weighted score than in more compact tasks. This explains why UC1 shows the largest difference between the two paradigms: 96.2% for the parametric framework compared with 77.7% for the autonomous baseline.

UC2 was less affected because the task required fewer retrieval steps and a more constrained output structure. Both approaches achieved 100.0% accuracy for this use case, indicating that autonomous orchestration can remain reliable when the retrieval path is short and the evidence requirements are explicit.

Overall, the robustness evaluation supports the use of constrained orchestration for the selected vulnerability-management benchmark. The benefit is not that deterministic workflows prevent all errors, but that they reduce the probability of cascading failures. In regulated financial-security operations, this matters because partial retrieval and unsupported synthesis can increase analyst review effort and lead to inappropriate remediation priorities.

6.6. Financial Impact Analysis

The preceding sections assessed accuracy, efficiency, and robustness. This section evaluates whether the technical differences between the two orchestration paradigms translate into estimated operational cost differences. The analysis separates direct model-use costs from indirect human-review costs. This distinction is necessary because token costs are likely to be small compared with the analyst effort required to review or correct incomplete outputs.

The first cost component is the token consumption. As shown in Section 6.4, the autonomous baseline required more prompt and completion tokens than the parametric approach. For the complete benchmark, the parametric approach required 3,840 tokens, compared with 4,960 tokens for the autonomous baseline. This represents a 29.2% overhead in token consumption for the autonomous baseline.

Using the cost model defined in Equation 3.4, the per-query cost can be projected to larger workloads. The annualized cost projections for the varying daily query rates are shown in Table 6.7. These projections reflect the same per-query token requirements as the benchmark and result in a linear extrapolation to 365 days.

Table 6.7: Projected annual direct token cost under different daily query volumes. The autonomous baseline is consistently more expensive because it consumes more tokens per question, but the absolute direct-token-cost difference remains relatively small.

Scenario	Daily queries	Parametric	Autonomous
Small SOC	500	$531	$686
Medium SOC	1,000	$1,061	$1,371
Large SOC	2,000	$2,122	$2,743
Tier-1 bank	5,000	$5,306	$6,857

Direct token-cost projections demonstrate that token savings are not the primary economic justification for the constrained orchestration. Even at 5000 daily queries, the resulting annual difference in direct token costs is substantially less than the costs of staffing and other operational activities. The main economic difference between the two approaches is not the direct token cost but the estimated correction cost associated with incomplete or unsupported outputs. Because the autonomous baseline produces more of such outputs under stochastic noise, its expected human-review cost is higher.

The second component of cost is human review. Incorrect or incomplete results in vulnerability management workflows do not simply vanish; instead, they must be examined, corrected, or rerun by an analyst. The results confirm that the autonomous baseline is more likely to require manual correction because it generates more incomplete results in the presence of stochastic noise. For this reason, the review-cost model estimates the expected proportion of outputs requiring manual correction.

In addition to the direct costs of review, incomplete outputs result in opportunity costs. The time spent revising the generated results represents the time lost for higher-value activities such as threat hunting, exposure analysis, coordination of remediation activities, or validation of critical exceptions to security controls. Therefore, the value achieved by constrained orchestration is not limited to reduced token consumption. Rather, it primarily derives from the reduced estimated review burden due to incomplete, or unsupported results.

The projected review rates are not intended to reflect the measured rates of production correction. Rather, they represent scenario values that illustrate the potential effects of variations in benchmark completeness and unsupported synthesis on analyst review requirements at the operational level.

Table 6.8 presents a simple annual human-review projection for a medium-size SOC processing 1,000 queries per day. The model reflects a manual review cost of $20 per correction. Thus, even small variations in the rate of human review result in large annual cost variations.

Table 6.8: Projected annual human-review cost for a medium-size SOC processing 1,000 queries per day. The cost difference is driven by the higher expected correction burden of the autonomous baseline.

Approach	Assumed review rate	Annual review cost
Parametric orchestration	0.4%	$29,200
Autonomous baseline	5.0%	$365,000

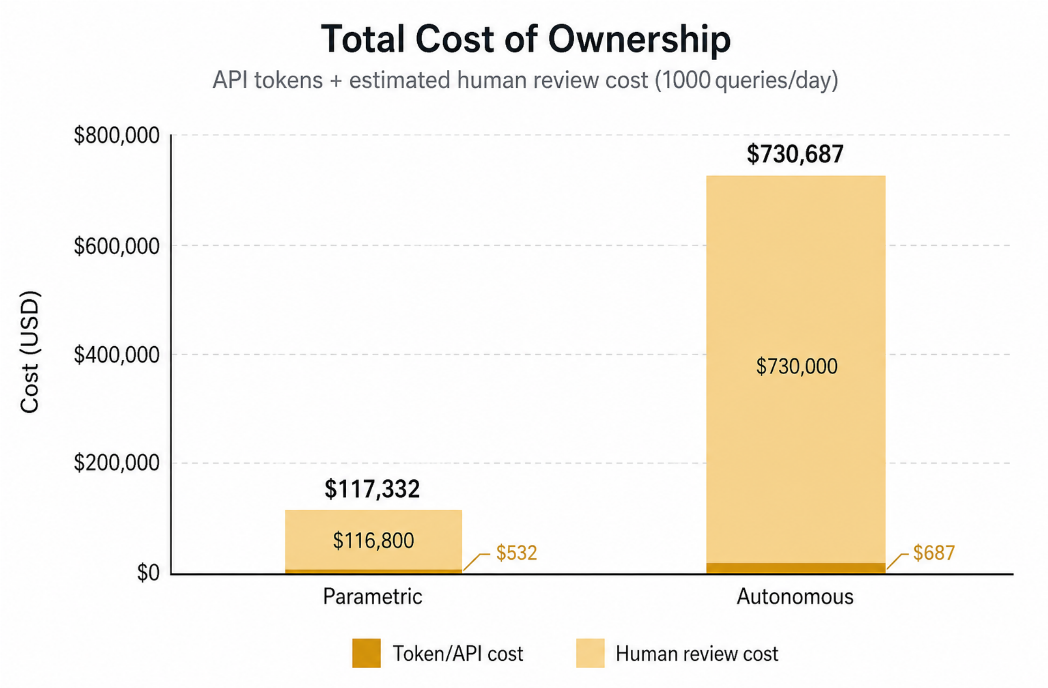

Figure 6.5: Projected annual total cost of ownership for a medium-size SOC processing 1,000 queries per day. The difference is dominated by estimated human-review cost rather than direct token cost.

The human-review projection illustrates why robustness matters operationally. The difference in direct token cost is measurable but relatively small, whereas the estimated cost of manually reviewing incomplete outputs can become substantial at scale. This is particularly relevant in financial-security environments, where unverified recommendations may introduce operational or regulatory risk.

The financial impact analysis therefore reinforces the architectural conclusion of this chapter. Parametric orchestration reduces token consumption, but its more important benefit is the reduction of partial retrieval and unsupported synthesis. For high-risk vulnerability-management workflows, reducing the estimated correction burden may provide greater operational value than direct model-cost savings alone.

6.7. Qualitative Error Analysis

The quantitative results show that the parametric framework outperformed the autonomous baseline in overall accuracy, efficiency, and robustness on the selected benchmark. The qualitative error analysis explains why these differences occurred. The observed failures mainly concerned the construction of the evidence base before synthesis: some required retrieval steps were missed, some expected fields were omitted, and some final responses included claims that were not supported by the retrieved context.

The most important failure mode was incomplete multi-hop retrieval, especially in UC1. Contextual asset triage requires the system to retrieve a CVE, identify all affected CMDB assets, and enrich each asset with ownership and NSF compliance context. The parametric workflow encodes this sequence explicitly. The autonomous baseline occasionally omitted one of these steps or failed to integrate all affected assets, which directly reduced the server, owner, and supporting-field components of the weighted accuracy score.

A second failure mode was unsupported synthesis. In some cases, the autonomous baseline produced plausible but unsupported information, such as invented owner names or compliance values. In vulnerability prioritization, this type of error is more serious than explicitly reporting missing information. An analyst can act appropriately when uncertainty is visible, whereas fabricated context may appear authoritative and distort the prioritization decision.

A third failure mode was unnecessary tool exploration. The autonomous baseline sometimes invoked tools that were not required for the selected workflow. This did not always reduce accuracy, but it increased token consumption and made the execution trace less predictable. Such exploration may be useful in open-ended investigations, but it is inefficient when the required retrieval path is already known.

A final failure mode concerned incomplete structured outputs. This occurred mainly in remediation-oriented tasks, where the response was generally useful but omitted a required element such as an owner, deadline, target host, or indicator of compromise. These omissions did not always invalidate the response, but they reduced its actionability for analysts.

Table 6.9 summarizes the qualitative error taxonomy used in the benchmark.

Table 6.9: Qualitative error taxonomy observed in the benchmark. The dominant autonomous-agent failures are associated with incomplete retrieval, unsupported synthesis, and unnecessary exploration.

Error type	Description	Architectural implication
Missed retrieval hop	A required retrieval step, such as a compliance lookup, is skipped.	Required multi-hop workflows should be encoded deterministically.
Dropped asset	One or more expected affected assets are omitted from the response.	Asset aggregation should be handled using programmatic retrieval logic.
Hallucinated field	A plausible but unsupported owner, compliance score, or contextual claim is generated.	Generated claims should be grounded in retrieved context and validated.
Redundant tool call	The agent performs exploratory calls that are not required for the selected workflow.	Tool access should be restricted when the workflow is known.
Missing output field	A required section or value is absent from the final response.	Output schemas and required-field checks should be applied before returning results.

The benefit of the parametric framework was therefore not that it generated more sophisticated final language. When both paradigms retrieved the same evidence, their final responses were broadly comparable in content and organization. The main difference was architectural: the parametric framework made it harder to omit necessary retrieval steps, lose required fields, or synthesize claims without supporting evidence.

Taken together, the evaluation results indicate that the benefit of the parametric framework lies primarily in the reliability of the orchestration process. The framework does not make the LLM inherently better at reasoning about vulnerabilities; rather, it improves the conditions under which synthesis takes place by ensuring that relevant evidence is retrieved, reduced, and validated before the response is generated. In the selected controlled benchmark, the parametric approach performed better because the tasks were structured enough for predefined workflows to capture the required evidence chains. The results therefore support constrained orchestration for high-risk vulnerability-prioritization tasks, while leaving open the possibility that more autonomous approaches may be useful in exploratory settings where the relevant retrieval path is not known in advance.

7 Discussion

This chapter interprets the evaluation results and discusses their implications for LLM-supported vulnerability prioritization in financial-security operations. It explains why the parametric framework performed better on the selected benchmark, identifies the operational meaning of these results, and defines the appropriate automation boundary for the proposed system. The chapter also discusses threats to validity, limitations, and directions for future work.

7.1. Interpretation of Findings

The evaluation results support the use of constrained orchestration for the structured vulnerability-management workflows considered in this thesis. The parametric framework achieved higher overall accuracy, consumed fewer tokens, and degraded less under stochastic workflow noise than the autonomous-agent baseline. These results do not show that parametric orchestration is universally preferable to autonomous orchestration. Rather, they show that deterministic control is beneficial when the task requires known evidence, predictable joins, and complete retrieval before synthesis.

The strongest evidence for this interpretation comes from contextual asset triage. This use case requires a dependent retrieval chain: the system must begin with a CVE, identify affected CMDB assets, retrieve ownership information, and enrich the result with compliance context. If one of these steps is omitted, the final response may still appear plausible while missing information that is essential for prioritization. The parametric framework reduces this risk because the retrieval sequence is encoded directly in the workflow. The autonomous baseline, by contrast, must decide during execution which steps are necessary, which creates more opportunities for incomplete evidence construction.

The risk-score justification results show that autonomy is not inherently unsuitable. Both paradigms achieved equivalent accuracy for this use case because the required evidence was compact and could be retrieved through a short, structured path. However, the autonomous baseline required more tool calls and tokens to reach the same outcome. This indicates that autonomous orchestration can perform well for low-hop tasks, but it does not necessarily improve efficiency when the required workflow is already known.

The remediation-guidance use case occupies an intermediate position. It requires more contextual information than risk-score justification but less dependent retrieval than contextual asset triage. The autonomous baseline performed close to the parametric framework, with only minor omissions. This supports the broader interpretation that the advantage of deterministic orchestration increases as the number of dependent retrieval steps and required output fields increases.

The robustness results reinforce this interpretation. The parametric framework was less vulnerable to stochastic workflow noise because predefined templates continued to attempt the required retrieval operations even when individual fields or records were affected. In the autonomous baseline, an early planning or retrieval error could remove evidence needed for later parts of the answer. This cascading effect is particularly important in vulnerability prioritization, where missing ownership, exposure, or compliance context can change how analysts interpret urgency and remediation responsibility.

Overall, the main benefit of the proposed framework is not that it makes the LLM intrinsically better at vulnerability reasoning. Instead, it improves the conditions under which LLM-based synthesis takes place. By retrieving evidence deterministically, pruning context before generation, and validating outputs after generation, the framework reduces the probability that the model will produce explanations based on incomplete or unsupported data.

7.2. Implications for Financial Security Operations

The results suggest that LLM-supported systems for financial-security operations should be designed as controlled decision-support tools rather than general-purpose chat interfaces. Vulnerability prioritization is not an open-ended conversation: it usually requires specific evidence about affected assets, ownership, exposure, compliance posture, and remediation constraints. A workflow that retrieves and structures this evidence before synthesis is therefore more appropriate than an interface that leaves factual discovery to the model.

The results also show that completeness and traceability are more important than raw automation. In a financial setting, incomplete outputs still require analyst intervention, and unsupported outputs may introduce operational risk. The financial-impact analysis in Chapter 6 shows that estimated review costs can exceed direct token costs. The practical value of constrained orchestration therefore lies less in reducing model-use cost and more in reducing incomplete or unsupported outputs that require correction.

The framework also supports auditability. Because the parametric architecture separates retrieval operations, intermediate context, and generated output, analysts can inspect how a recommendation was produced. This is important in regulated environments, where security decisions must often be justified after the fact. A recommendation supported by explicit CVE records, asset records, ownership information, and compliance status is more defensible than a conclusion generated without an inspectable evidence chain.

The findings further support decision support rather than fully autonomous remediation. The framework can collect context, explain prioritization factors, and generate remediation-oriented summaries, but it does not apply patches, modify configurations, or initiate response actions. This boundary is appropriate because remediation decisions often involve operational trade-offs, such as service continuity, maintenance windows, ownership responsibilities, and residual risk acceptance.

Finally, the results support selective rather than unrestricted autonomy. Autonomous agents may remain useful for exploratory investigations where relevant sources or retrieval paths are not known in advance. For high-risk, well-established workflows with predictable evidence requirements, however, restricted orchestration provides stronger reliability, auditability, and control. A practical deployment could therefore combine both modes: parametric execution for structured vulnerability-management tasks and limited autonomous exploration for lower-risk investigative work.

Table 7.1: Operational implications of the evaluation findings.

Finding	Implication for financial security operations
Parametric orchestration improves multi-hop retrieval reliability	Structured vulnerability workflows should explicitly encode the required retrieval steps.
Autonomous agents introduce additional tool-call and token overhead	Flexibility should be reserved for tasks where exploration is necessary.
Noise affects autonomous workflows more strongly	High-stakes workflows should include deterministic retrieval and validation safeguards.
Human review cost can exceed direct token cost	Accuracy and completeness should be prioritized over raw automation.
Generated explanations are useful when grounded in retrieved context	LLMs are appropriate for analyst-facing synthesis, but not for unconstrained factual discovery.

7.3. Automation Boundary

The results support the use of LLMs for decision support, but not for fully autonomous vulnerability remediation. The framework can identify relevant context, summarize affected assets, justify priorities, and generate remediation guidance. However, these outputs remain inputs to analyst judgment rather than final operational decisions.

Analyst oversight is necessary at several points in the workflow. Analysts can inspect the retrieved CVE, CMDB, ownership, and compliance evidence before accepting a recommendation. They can also reject or qualify outputs when the retrieved information is incomplete, inconsistent, or insufficiently supported. Finally, they remain responsible for deciding whether remediation should be accelerated, postponed, or accepted as residual risk. This proof of concept does not include a learning mechanism based on analyst corrections, which remains a direction for future work.

This boundary is necessary because remediation decisions cannot be reduced to technical evidence alone. Applying a patch may require downtime, affect dependent services, conflict with maintenance windows, or require approval from service owners. In regulated financial environments, these decisions must remain accountable to human analysts and responsible operational teams.

The framework therefore automates information retrieval, context construction, and analyst-facing synthesis, but not operational execution. It can reduce the manual effort required to collect evidence and prepare explanations, while preserving human responsibility for final prioritization and remediation decisions.

Table 7.2: Automation boundary of the proposed framework.

Appropriate for automation	Requires human decision or approval
Retrieving CVE, CMDB, ownership, and compliance context	Applying patches or changing production configurations
Summarizing affected assets and contextual risk factors	Accepting or rejecting operational risk
Generating analyst-facing risk justifications	Approving remediation timelines or maintenance-window changes
Drafting remediation guidance and hunting-query suggestions	Executing response actions in production systems
Highlighting missing or inconsistent context	Resolving conflicting business priorities between teams

This boundary also defines the role of the LLM. The model is appropriate for synthesis, explanation, and communication, but not for controlling operational actions. It is therefore incorporated as part of a controlled decision-support process rather than as an independent security operator.

7.4. Threats to Validity

The evaluation provides evidence that constrained orchestration improves the reliability and efficiency of the selected structured vulnerability-management workflows. However, the results must be interpreted in light of several threats to validity. This section discusses internal, external, construct, and conclusion validity, following common categories from experimental design and empirical software engineering [54], [55], [59].

7.4.1. Internal Validity

Internal validity concerns whether the observed differences between the parametric framework and the autonomous baseline are caused by orchestration strategy rather than unrelated implementation choices. This threat was reduced by applying both paradigms to the same benchmark questions, data sources, model deployment, and tool interface. The main difference between the systems was therefore the degree of control over retrieval order, context construction, and tool use.

Nevertheless, the results depend on the specific workflow templates, tool definitions, prompt templates, and validation logic implemented in the proof of concept. A different implementation of either paradigm could change the absolute scores. The comparison should therefore be interpreted as evidence for the tested architecture and benchmark design, not as a universal measurement of all possible parametric or autonomous systems.

The stochastic noise model introduces an additional internal-validity concern. The injected errors model workflow-level failures such as missed retrieval hops, dropped records, incorrect parameters, redundant tool calls, hallucinated lines, and missing fields. These probabilities are modeling assumptions rather than empirical measurements of production failure rates. To mitigate this limitation, the evaluation included a sensitivity analysis across three noise scales and five random seeds. The parametric framework retained higher mean weighted accuracy across all tested settings, but future work should calibrate these probabilities using logs from real LLM-supported security workflows.

7.4.2. External Validity

External validity concerns whether the findings generalize beyond the experimental setting. The benchmark uses an approved static production-derived CSV snapshot of selected vulnerability-management, configuration-management, and compliance records. This makes the evaluation more realistic than a fully synthetic benchmark because it preserves operational relationships between vulnerabilities, assets, ownership, and compliance context.

At the same time, the snapshot is limited in scale, dynamism, and operational complexity. It is correct for the selected use cases at the time of export, but it is not a live connection to production databases. Changes in ownership, exposure, patch status, compliance posture, or service dependencies after the export are therefore not represented. Identifying fields were also masked or generalized to protect confidentiality and security, which limits the extent to which examples can be interpreted as direct representations of production conditions.

The findings should therefore be interpreted as proof-of-concept evidence from a controlled, production-derived benchmark. Broader generalization would require evaluation on larger datasets, more diverse vulnerability classes, more complex asset dependencies, and realistic production data-quality problems.

7.4.3. Construct Validity

Construct validity concerns whether the evaluation measures capture the concepts they are intended to measure. The weighted accuracy score captures three important components of vulnerability-prioritization output: correct identification of affected servers, correct attribution of ownership, and inclusion of expected supporting fields. These elements are operationally important because an output that omits affected assets, responsible owners, or key context is difficult for analysts to act on.

However, the metric does not fully capture all dimensions of response quality. It does not directly measure explanation clarity, analyst trust, cognitive workload, or the likelihood that a generated recommendation would change a real remediation decision. These aspects were considered only indirectly through qualitative interpretation. Future evaluations should include expert ratings or user studies to assess perceived usefulness, readability, and operational value.

The scoring method also treats the loss of different fields in a simplified way. In practice, omitting a critical production asset may be more serious than omitting a lower-risk contextual field. Future work should therefore consider risk-weighted or penalty-based scoring, where errors involving critical assets, high exposure, or regulatory relevance have stronger effects on the final score.

The financial impact model is also simplified. It estimates direct token costs and expected human-review costs, but it does not include integration effort, monitoring, incident escalation, infrastructure overhead, model governance, or long-term maintenance. The model is therefore useful for identifying relative cost drivers, but not for producing a complete production business case.

7.4.4. Conclusion Validity

Conclusion validity concerns whether the conclusions drawn from the data are sufficiently supported. The results show that the parametric framework outperformed the autonomous baseline on the selected benchmark. However, the benchmark is small and focuses on structured workflows with known data sources and predictable retrieval paths. The conclusions should therefore be limited to similar vulnerability-management conditions.

The results do not establish that autonomous agents are generally inferior. Autonomous orchestration may be better suited to exploratory investigations, ill-defined tasks, or workflows in which the relevant sources are not known in advance. The supported conclusion is narrower: for structured, high-risk vulnerability-prioritization tasks in regulated financial environments, deterministic orchestration provides stronger reliability, traceability, and control than unconstrained autonomous tool use.

7.5. Limitations

The findings of this thesis demonstrate architectural feasibility, not full production generalizability. The benchmark was deliberately small, static, and controlled in order to isolate the effect of orchestration strategy. The results therefore show how the proposed framework behaves on representative vulnerability-management tasks, but they do not establish that the same accuracy, efficiency, or robustness would be achieved across all production assets, vulnerability classes, data-quality conditions, or analyst questions.

The first limitation is the use of a static offline dataset. The evaluation is based on an approved static production-derived CSV snapshot of selected vulnerability-management, configuration-management, and compliance systems. The snapshot preserves the relationships required for the selected use cases, but it does not include real-time changes in ownership, exposure, patch status, compliance posture, or operational dependencies.

The second limitation concerns scale and coverage. The approved subset contains the records required for the benchmark, but it does not represent the full complexity of a large financial institution. A production implementation would need to support larger asset inventories, overlapping vulnerabilities, more diverse asset classes, complex service dependencies, and inconsistent or incomplete data.

The third limitation is model dependence. The evaluation used a bank-approved offline deployment of Claude Opus 4.6 with minimal task-specific adaptation. This provides a realistic basis for internal experimentation, but the results may differ for other model families, versions, deployment conditions, prompt templates, or fine-tuning strategies.

The fourth limitation concerns the stochastic noise model. The injected errors represent plausible workflow-level failures for LLM-supported systems, but their probabilities are experimental assumptions. Alternative noise profiles would change the absolute performance values, even if the relative robustness advantage of constrained orchestration remained.

The fifth limitation is the absence of an analyst user study. The benchmark measures accuracy, tool use, token consumption, robustness, and estimated review cost, but it does not directly measure perceived usefulness, trust, cognitive workload, or acceptance by security analysts. These factors are essential for assessing operational deployment.

Finally, the framework was evaluated as a proof of concept rather than as a live production system. It was not connected to live databases, scanners, ticketing systems, remediation workflows, or production servers during benchmark execution. Live deployment would require additional mechanisms for authentication, authorization, monitoring, incident escalation, service-level management, change-management integration, continuous data-quality monitoring, and continuous model-quality control.

7.6. Future Work

Future work should first evaluate the framework on larger and more dynamic enterprise datasets. The current evaluation uses an approved static production-derived CSV snapshot of selected vulnerability, asset, and compliance records. Subsequent work should test the framework on larger approved datasets and, where feasible, controlled live integrations with enterprise data sources. This would allow evaluation of scalability, retrieval latency, access-control restrictions, data freshness, and changing asset relationships under more realistic operational conditions.

A related direction is to extend the enterprise context used for prioritization. Future versions of the framework could incorporate software supply-chain and supplier-risk information, including product dependencies, third-party ownership, software bills of materials, CSAF advisories, VEX status information, and indicators of supplier risk. This would help determine whether a vulnerability is relevant to a specific product, dependency, or deployment mode in a given operational setting [60], [61], [62].

A second direction is to evaluate alternative model backends and deployment configurations. The current proof of concept uses a bank-approved offline deployment of Claude Opus 4.6. Future studies should compare other private, on-premises, or offline model deployments and quantify model variability, latency, prompt sensitivity, consistency across repeated runs, and the trade-off between fine-tuning and retrieval-based prompting.

A third direction is more realistic robustness testing. Future work should calibrate the stochastic noise model using logs from real LLM-supported security workflows and evaluate additional enterprise data-quality problems. Examples include outdated ownership records, conflicting asset classifications, missing compliance records, duplicate assets, outdated patch information, and inconsistent asset-naming conventions. This would provide a more complete understanding of how the framework behaves under realistic operational uncertainty.

A fourth direction is analyst-centered evaluation. User studies with security analysts should assess whether the generated explanations are useful in practice. Such studies could measure perceived trust, clarity, time savings, cognitive workload, and impact on remediation decisions. They could also examine how analyst corrections should be incorporated into workflow templates, validation rules, and context-selection strategies.

A fifth direction is stronger validation of generated output. The proof of concept includes basic checks for field completeness and structural consistency, but future work should add citation-based grounding, stricter schema validation, policy-rule validation, and automated detection of unsupported assertions. Remediation-oriented workflows should also include more rigorous validation of generated KQL or other security query languages.

A final direction is hybrid orchestration. The results support parametric orchestration for known structured tasks while preserving a role for autonomous agents in exploratory work. A future system could combine both modes by using parametric execution for high-risk repetitive workflows and bounded autonomous exploration for lower-risk investigations. The main challenge is to define safe boundaries between these modes, including fallback rules for uncertainty, missing evidence, or unexpected tool behavior.

Future production integration should maintain the automation boundary established in this thesis. The system may support prioritization and remediation planning, but final operational decisions should remain under human control.

8 Conclusion

This chapter concludes the thesis by summarizing the main findings, answering the research questions, restating the contributions, and outlining the most important directions for future work. It closes by reflecting on the role of context-aware orchestration for LLM-supported vulnerability prioritization in regulated financial environments.

8.1. Summary

This thesis designed, implemented, and evaluated a context-aware orchestration framework for LLM-supported vulnerability prioritization. The framework addresses a central challenge in financial-security operations: public vulnerability indicators such as CVSS, EPSS, and known-exploitation evidence are useful, but they are insufficient without organization-specific context about affected assets, ownership, exposure, compliance posture, and remediation constraints.

The proposed framework separates deterministic enterprise-data retrieval from LLM-based synthesis. Factual evidence is retrieved through predefined workflows and schema-aware joins, reduced through metadata pruning, and validated before being used to generate analyst-facing explanations. The framework is therefore intended as a decision-support system, not as an autonomous remediation system.

The proof-of-concept was evaluated in a controlled offline environment using an approved static production-derived CSV snapshot from selected operational vulnerability-management, configuration-management, and compliance systems. Across a 14-question benchmark covering contextual asset triage, risk-score justification, and remediation guidance, the parametric framework achieved higher accuracy, lower token consumption, and stronger robustness under stochastic workflow noise than the autonomous-agent baseline. These results support constrained orchestration as a promising architectural pattern for structured vulnerability-management workflows where retrieval completeness, traceability, and validation are critical.

8.2. Research Questions Revisited

This section answers the research questions introduced in Chapter 1.

RQ1: How can heterogeneous security and organizational data be selectively retrieved and transformed into context usable for vulnerability prioritization?

Heterogeneous security and organizational data can be transformed into usable prioritization context through deterministic retrieval, schema-aware joins, and task-specific context construction. In the proposed framework, vulnerability records, CMDB asset metadata, ownership information, and compliance records are accessed through constrained retrieval functions rather than through unrestricted LLM tool use. Explicit relationships, such as the mapping between vulnerabilities and affected assets and between assets and compliance records, provide the basis for multi-hop retrieval.

The retrieved records are then reduced to task-specific context objects. These objects retain the fields required for the selected workflow, such as affected assets, owners, classifications, compliance status, risk level, patch information, and remediation constraints. Irrelevant metadata are removed before LLM-based synthesis, reducing token consumption and limiting unnecessary exposure of internal operational information. This approach gives the model structured evidence for explanation while preserving deterministic control over factual retrieval.

RQ2: How well does the proposed orchestration framework perform on representative vulnerability-management tasks in terms of accuracy, operational overhead, estimated correction effort, and operational cost?

On the selected benchmark, the parametric orchestration framework achieved higher accuracy, lower operational overhead, and stronger robustness than the autonomous-agent baseline. The framework achieved 98.4% mean weighted accuracy, compared with 90.0% for the autonomous baseline. The largest difference occurred in contextual asset triage, where the parametric framework achieved 96.2% accuracy compared with 77.7% for the autonomous baseline. This result supports the value of deterministic orchestration for multi-hop retrieval tasks, where omitting one retrieval step can remove important downstream evidence.

The parametric framework also consumed fewer language-model tokens. Across the benchmark, it required 3,840 tokens compared with 4,960 tokens for the autonomous baseline, corresponding to 29.2% higher token use by the autonomous approach. The financial-impact analysis further suggests that the main operational cost difference is not direct token cost, but the estimated correction burden associated with incomplete or unsupported outputs. Accuracy, completeness, and robustness are therefore more important cost drivers than token consumption alone in the evaluated setting.

RQ3: Which architectural safeguards are required to make LLM-supported vulnerability prioritization usable in a regulated financial environment?

The evaluation indicates that LLM-supported vulnerability prioritization requires safeguards at the level of data access, context construction, output validation, and operational control. First, factual retrieval should be deterministic and traceable. The LLM should not be responsible for discovering database schemas, selecting arbitrary data sources, or deciding whether critical retrieval steps are necessary.

Second, the system should apply metadata pruning before LLM-based synthesis. Only task-relevant context should be passed to the model, which improves efficiency and limits unnecessary exposure of sensitive internal information. Third, generated outputs should be validated before being returned to analysts. Required fields, grounding against retrieved evidence, output structure, and query syntax should be checked where applicable.

Finally, the framework should preserve a clear automation boundary. It may support triage, risk justification, remediation planning, and analyst communication, but it should not apply patches, modify configurations, or initiate production changes. These safeguards support the control, accountability, and human oversight expected in regulated financial operations [2], [12], [43].

8.3. Contributions

This thesis makes four main contributions to the design and evaluation of governance-aware LLM orchestration for vulnerability prioritization.

First, it presents a context-aware orchestration framework that combines deterministic enterprise-data retrieval with bounded LLM-based synthesis. The framework establishes an evidence boundary before generation by separating retrieval, schema-aware joining, metadata pruning, validation, and analyst-facing explanation.

Second, it defines a task-specific context-construction approach for integrating heterogeneous vulnerability, asset, ownership, exposure, and compliance information. This approach reduces unnecessary metadata while preserving the relationships required to support prioritization decisions.

Third, it implements and evaluates a proof-of-concept system across three representative vulnerability-management use cases: contextual asset triage, risk-score justification, and remediation guidance.

The evaluation shows that constrained orchestration improves accuracy and robustness over the autonomous-agent baseline for the structured workflows in the selected benchmark.

Fourth, it identifies architectural safeguards for LLM-supported vulnerability prioritization in regulated financial settings. These safeguards include controlled data access, deterministic retrieval, metadata pruning, output validation, evidence traceability, and preservation of analyst oversight.

8.4. Future Work

Future work should evaluate the framework on larger, more dynamic, and more diverse enterprise datasets. Controlled live integrations with vulnerability scanners, CMDB systems, SIEM platforms, ticketing systems, and compliance tools would allow assessment of scalability, latency, data freshness, access control, and production data quality.

Further research should also compare alternative private or on-premises model deployments, strengthen output validation, and evaluate the framework with security analysts. Analyst-centered studies are particularly important because this thesis measures technical output quality, but does not directly assess perceived usefulness, trust, workload reduction, or acceptance in routine operations.

Finally, future versions of the framework could incorporate additional enterprise context, including software supply-chain information, SBOM data, supplier-risk indicators, and structured vulnerability status formats such as CSAF or VEX, as discussed in Section 7.6.

8.5. Final Remarks

The principal conclusion of this thesis is that large language models can support vulnerability prioritization in financial infrastructure, but only when they are embedded within a controlled architecture. The value of such a system depends not only on the quality of generated explanations, but also on the reliability of evidence retrieval, the traceability of intermediate steps, the protection of sensitive information, and the preservation of human oversight.

The results show that constrained, context-aware orchestration is a promising architectural pattern for structured vulnerability-management tasks. By separating deterministic retrieval and validation from LLM-based synthesis, the framework provides analyst-facing decision support without transferring control over data access, evidence selection, or operational action to the model.

The findings should be interpreted as proof-of-concept evidence over representative use cases and an approved static production-derived CSV snapshot. Deployment in live financial infrastructure would require further validation on larger and dynamic datasets, integration with production systems, access-control mechanisms, monitoring, governance review, and analyst-centered evaluation. Nevertheless, the study demonstrates that careful orchestration can make LLM-supported vulnerability prioritization more grounded, traceable, and operationally useful in regulated financial-security settings.

References

[1] M. Souppaya and K. Scarfone, “Guide to Enterprise Patch Management Planning: Preventive Maintenance for Technology,” National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-40 Revision 4, 2022. Accessed: May 1, 2026. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/40/r4/final.

[2] European Union. “Regulation (EU) 2022/2554 on Digital Operational Resilience for the Financial Sector,” Accessed: May 1, 2026. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2022/2554/oj.

[3] European Union Agency for Cybersecurity, “ENISA Threat Landscape 2024,” European Union Agency for Cybersecurity, Tech. Rep., 2024. DOI: 10.2824/0710888. Accessed: May 4, 2026. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024.

[4] European Union Agency for Cybersecurity, “ENISA Threat Landscape: Finance Sector,” ENISA, Tech. Rep., 2025. Accessed: May 11, 2026. [Online]. Available: https://www.enisa.europa.eu/sites/default/files/2025-02/Finance%20TL%202024_Final.pdf.

[5] Forum of Incident Response and Security Teams. “Common Vulnerability Scoring System Version 4.0: Specification Document,” Accessed: May 1, 2026. [Online]. Available: https://www.first.org/cvss/v4.0/specification-document.

[6] Forum of Incident Response and Security Teams. “Exploit Prediction Scoring System (EPSS),” Accessed: May 1, 2026. [Online]. Available: https://www.first.org/epss/.

[7] Cybersecurity and Infrastructure Security Agency. “Known Exploited Vulnerabilities Catalog,” Accessed: May 1, 2026. [Online]. Available: https://www.cisa.gov/known-exploited-vulnerabilities-catalog.

[8] J. M. Spring et al., “Prioritizing Vulnerability Response: A Stakeholder-Specific Vulnerability Categorization, Version 2.0,” Carnegie Mellon University Software Engineering Institute, Tech. Rep., 2021. Accessed: May 1, 2026. [Online]. Available: https://www.sei.cmu.edu/library/prioritizing-vulnerability-response-a-stakeholder-specific-vulnerability-categorization-version-20/.

[9] Cybersecurity and Infrastructure Security Agency. “Stakeholder-Specific Vulnerability Categorization (SSVC),” Accessed: May 1, 2026. [Online]. Available: https://www.cisa.gov/stakeholder-specific-vulnerability-categorization-ssvc.

[10] M. T. Alam, D. Bhusal, L. Nguyen, and N. Rastogi, “CTIBench: A Benchmark for Evaluating LLMs in Cyber Threat Intelligence,” in *Advances in Neural Information Processing Systems*, vol. 37, 2024. Accessed: May 1, 2026. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/hash/5acd3c628aa1819fbf07c39ef73e7285-Abstract-Datasets_and_Benchmarks_Track.html.

[11] L. Huang et al., “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions,” *arXiv preprint arXiv:2311.05232*, 2023. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2311.05232.

[12] National Institute of Standards and Technology, “Artificial Intelligence Risk Management Framework (AI RMF 1.0),” National Institute of Standards and Technology, Tech. Rep. NIST AI 100-1, 2023. DOI: 10.6028/NIST.AI.100-1. Accessed: May 1, 2026. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf.

[13] N. F. Liu et al., “Lost in the Middle: How Language Models Use Long Contexts,” *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024. DOI: 10.1162/tacl_a_00638. Accessed: May 1, 2026. [Online]. Available: https://aclanthology.org/2024.tacl-1.9/.

[14] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9459–9474. Accessed: May 1, 2026. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

[15] Y. Gao et al., “Retrieval-Augmented Generation for Large Language Models: A Survey,” *arXiv preprint arXiv:2312.10997*, 2023. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2312.10997.

[16] T. Schick et al., “Toolformer: Language Models Can Teach Themselves to Use Tools,” in *Advances in Neural Information Processing Systems*, 2023. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2302.04761.

[17] S. Yao et al., “ReAct: Synergizing Reasoning and Acting in Language Models,” in *International Conference on Learning Representations*, 2023. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2210.03629.

[18] National Institute of Standards and Technology, “The NIST Cybersecurity Framework (CSF) 2.0,” National Institute of Standards and Technology, Tech. Rep. NIST Cybersecurity White Paper, 2024. DOI: 10.6028/NIST.CSWP.29. Accessed: May 1, 2026. [Online]. Available: https://www.nist.gov/cyberframework.

[19] The CVE Program. “Common Vulnerabilities and Exposures (CVE),” Accessed: May 1, 2026. [Online]. Available: https://www.cve.org/.

[20] Forum of Incident Response and Security Teams. “Common Vulnerability Scoring System Version 4.0: User Guide,” Accessed: May 1, 2026. [Online]. Available: https://www.first.org/cvss/user-guide.

[21] National Vulnerability Database. “Vulnerability Metrics: Common Vulnerability Scoring System,” Accessed: May 1, 2026. [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss.

[22] J. Jacobs, S. Romanosky, B. Edwards, M. Roytman, and I. Adjerid, “Exploit Prediction Scoring System (EPSS),” *ACM Digital Threats: Research and Practice*, vol. 2, no. 3, 2021. DOI: 10.1145/3436242. [Online]. Available: https://doi.org/10.1145/3436242.

[23] K. Dempsey et al., “Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations,” National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-137, 2011. DOI: 10.6028/NIST.SP.800-137. Accessed: May 4, 2026. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/137/final.

[24] MITRE. “MITRE ATT&CK Enterprise Matrix,” Accessed: May 1, 2026. [Online]. Available: https://attack.mitre.org/.

[25] C. Johnson, L. Badger, D. Waltermire, J. Snyder, and C. Skorupka, “Guide to Cyber Threat Information Sharing,” National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-150, 2016. DOI: 10.6028/NIST.SP.800-150. Accessed: May 1, 2026. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/150/final.

[26] Y. Jiang, N. Oo, Q. Meng, H. W. Lim, and B. Sikdar, “A Survey on Vulnerability Prioritization: Taxonomy, Metrics, and Research Challenges,” *arXiv preprint arXiv:2502.11070*, 2025. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2502.11070.

[27] B. Jung, Y. Li, and T. Bechor, “CAVP: A Context-Aware Vulnerability Prioritization Model,” *Computers & Security*, vol. 116, p. 102 639, 2022. DOI: 10.1016/j.cose.2022.102639. Accessed: May 1, 2026. [Online]. Available: https://doi.org/10.1016/j.cose.2022.102639.

[28] G. d. J. C. da Silva and C. B. Westphall, “A Survey of Large Language Models in Cybersecurity,” *arXiv preprint arXiv:2402.16968*, 2024. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2402.16968.

[29] H. Xu, G. Wang, Y. Li, L. Zhang, Y. Li, and Y. Liu, “Large Language Models for Cyber Security: A Systematic Literature Review,” *arXiv preprint arXiv:2405.04760*, 2024. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2405.04760.

[30] N. O. Jaffal, “Large Language Models in Cybersecurity,” *arXiv preprint arXiv:2507.13629*, 2025. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2507.13629.

[31] D. Bhusal et al., "SECURE: Benchmarking Large Language Models for Cybersecurity," *arXiv preprint arXiv:2405.20441*, 2024. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2405.20441.

[32] M. Bhatt, S. Chennabasappa, C. Nikolaidis, et al., "Purple Llama CyberSecEval: A Secure Coding Benchmark for Language Models," *arXiv preprint arXiv:2312.04724*, 2023. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2312.04724.

[33] Y. Sun et al., "LLM4Vuln: A Unified Evaluation Framework for Decoupling and Enhancing LLMs' Vulnerability Reasoning," *arXiv preprint arXiv:2401.16185*, 2024. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2401.16185.

[34] Y. Meng et al., "Uncovering Vulnerabilities of LLM-Assisted Cyber Threat Intelligence," *arXiv preprint arXiv:2509.23573*, 2025. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2509.23573.

[35] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated Evaluation of Retrieval Augmented Generation," in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, Association for Computational Linguistics, 2024, pp. 150–158. Accessed: May 1, 2026. [Online]. Available: https://aclanthology.org/2024.eacl-demo.16/.

[36] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, "ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2024, pp. 338–354. Accessed: May 1, 2026. [Online]. Available: https://aclanthology.org/2024.naacl-long.20/.

[37] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," in *International Conference on Learning Representations*, 2024. Accessed: May 1, 2026. [Online]. Available: https://openreview.net/forum?id=hSyW5go0v8.

[38] M. Li et al., "API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2023, pp. 3102–3116. Accessed: May 1, 2026. [Online]. Available: https://aclanthology.org/2023.emnlp-main.187/.

[39] S. G. Patil et al., "The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models," in *Proceedings of the 42nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 267, PMLR, 2025, pp. 48 371–48 392. Accessed: May 1, 2026. [Online]. Available: https://proceedings.mlr.press/v267/patil25a.html.

[40] Y. Qin et al., "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs," in *International Conference on Learning Representations*, 2024. Accessed: May 1, 2026. [Online]. Available: https://openreview.net/forum?id=dHng2O0Jjr.

[41] X. Liu et al., "AgentBench: Evaluating LLMs as Agents," *arXiv preprint arXiv:2308.03688*, 2023. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2308.03688.

[42] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large Language Model Connected with Massive APIs," in *Advances in Neural Information Processing Systems*, 2024. Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2305.15334.

[43] European Union. "Directive (EU) 2022/2555 on Measures for a High Common Level of Cybersecurity Across the Union," Accessed: May 1, 2026. [Online]. Available: https://eur-lex.europa.eu/eli/dir/2022/2555/oj.

[44] J. T. Force, "Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-37 Revision 2, 2018. DOI: 10.6028/NIST.SP.800-37r2. Accessed: May 4, 2026. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/37/r2/final.

[45] J. T. Force, “Security and Privacy Controls for Information Systems and Organizations,” National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-53 Revision 5, 2020. DOI: 10.6028/NIST.SP.800-53r5. Accessed: May 4, 2026. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final.

[46] Cybersecurity and Infrastructure Security Agency, “Zero Trust Maturity Model Version 2.0,” Cybersecurity and Infrastructure Security Agency, Tech. Rep., 2023. Accessed: May 4, 2026. [Online]. Available: https://www.cisa.gov/sites/default/files/2023-04/zero_trust_maturity_model_v2_508.pdf.

[47] European Union. “Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence,” Accessed: May 1, 2026. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2024/1689/oj.

[48] National Institute of Standards and Technology, “Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile,” National Institute of Standards and Technology, Tech. Rep. NIST AI 600-1, 2024. DOI: 10.6028/NIST.AI.600-1. Accessed: May 1, 2026. [Online]. Available: https://www.nist.gov/publications/artificial-intelligence-risk-management-framework-generative-artificial-intelligence-profile.

[49] European Banking Authority, “EBA Guidelines on ICT and Security Risk Management,” European Banking Authority, Tech. Rep. EBA/GL/2019/04, 2019. Accessed: May 1, 2026. [Online]. Available: https://www.eba.europa.eu/regulation-and-policy/internal-governance/guidelines-on-ict-and-security-risk-management.

[50] OWASP Foundation. “OWASP Top 10 for Large Language Model Applications,” Accessed: May 1, 2026. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/.

[51] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design Science in Information Systems Research,” *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004. Accessed: May 1, 2026. [Online]. Available: https://aisel.aisnet.org/misq/vol28/iss1/6/.

[52] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research,” *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. DOI: 10.2753/MIS0742-1222240302. Accessed: May 1, 2026. [Online]. Available: https://doi.org/10.2753/MIS0742-1222240302.

[53] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Berlin, Heidelberg: Springer, 2014. DOI: 10.1007/978-3-662-43839-8. Accessed: May 1, 2026. [Online]. Available: https://doi.org/10.1007/978-3-662-43839-8.

[54] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer, 2012. DOI: 10.1007/978-3-642-29044-2. Accessed: May 1, 2026. [Online]. Available: https://doi.org/10.1007/978-3-642-29044-2.

[55] W. R. Shadish, T. D. Cook, and D. T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Boston, MA: Houghton Mifflin, 2002, ISBN: 9780395615560.

[56] R. Anderson, “Why Information Security is Hard—An Economic Perspective,” in *Proceedings of the 17th Annual Computer Security Applications Conference*, IEEE Computer Society, 2001, pp. 358–365. DOI: 10.1109/ACSAC.2001.991552. Accessed: May 1, 2026. [Online]. Available: https://www.acsac.org/2001/papers/110.pdf.

[57] R. Anderson and T. Moore, “The Economics of Information Security,” *Science*, vol. 314, no. 5799, pp. 610–613, 2006. DOI: 10.1126/science.1130992. Accessed: May 1, 2026. [Online]. Available: https://www.science.org/doi/10.1126/science.1130992.

[58] L. A. Gordon and M. P. Loeb, “The Economics of Information Security Investment,” *ACM Transactions on Information and System Security*, vol. 5, no. 4, pp. 438–457, 2002. DOI: 10.1145/581271.581274. Accessed: May 1, 2026. [Online]. Available: https://doi.org/10.1145/581271.581274.

[59] P. Runeson and M. Höst, “Guidelines for Conducting and Reporting Case Study Research in Software Engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009. DOI: 10.1007/s10664-008-9102-8. Accessed: May 1, 2026. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8.

[60] J. Boyens, A. Smith, N. Bartol, K. Winkler, A. Holbrook, and M. Fallon, “Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations,” National Institute of Standards and Technology, NIST Special Publication 800-161 Rev. 1, Update 1, 2024. DOI: 10.6028/NIST.SP.800-161r1-upd1. Accessed: May 11, 2026. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/161/r1/upd1/final.

[61] OASIS Open, *Common Security Advisory Framework Version 2.0*, OASIS Standard, edited by Langley Rock, Stefan Hagen, and Thomas Schmidt, 2022. Accessed: May 11, 2026. [Online]. Available: https://docs.oasis-open.org/csaf/csaf/v2.0/os/csaf-v2.0-os.html.

[62] Cybersecurity and Infrastructure Security Agency, *Vulnerability Exploitability eXchange (VEX) Status Justification*, 2022. Accessed: May 11, 2026. [Online]. Available: https://www.cisa.gov/sites/default/files/publications/VEX_Status_Justification_Jun22.pdf.

A

Benchmark Questions, Ground Truth, and Scoring Methodology

This appendix documents the benchmark used to compute the weighted accuracy results reported in Chapter 6. It provides the benchmark questions, expected ground-truth elements, per-question weighted scores, and score components for the parametric framework and autonomous-agent baseline. The purpose is to make the evaluation auditable and to clarify how the aggregate accuracy values were obtained.

A.1. Scoring Methodology

Each benchmark response was scored using the weighted accuracy formula introduced in Section 3.2.3:

$$W_{\text{total}} = 0.35S_{\text{server}} + 0.35S_{\text{owner}} + 0.30S_{\text{field}}.$$

Here, S_{server} is the fraction of expected affected servers correctly identified, S_{owner} is the fraction of expected owners correctly identified, and S_{field} is the fraction of expected supporting fields included in the response. Matching was performed using a case-insensitive comparison against the expected hostnames, owner names, and required field indicators.

The scoring method is intentionally simple and auditable. It evaluates whether the response contains the expected operational evidence, but it does not measure stylistic quality, analyst trust, readability, or subjective usefulness. The weighted scores reported in this appendix are rounded to the nearest whole percentage point.

The weighting gives slightly greater importance to affected server and owner identification because these elements determine whether remediation can be assigned to the correct operational team. Supporting fields are also important for explanation and completeness, but missing one supporting field typically reduces answer quality rather than making the response entirely unusable.

A.2. Benchmark Ground Truth

Table A.1 lists the 14 benchmark questions and their expected ground truth elements.

Table A.1: Benchmark questions and expected ground-truth elements.

Q	UC	Benchmark question	Expected affected servers	Expected owners	Expected supporting fields
Q1	UC1	Who owns the server affected by CVE-2026-1001, and what is its current NSF compliance status?	prod-web-eu-01; prod-db-us-01; prod-pay-us-01	Maria Lopez; James Carter; Priya Sharma	compliance_score; compliance_status; risk_level
Q2	UC1	What is the compliance status of the server prod-web-eu-01?	prod-web-eu-01	Maria Lopez	compliance_score; compliance_status
Q3	UC1	List all servers with critical risk level and their owners.	dev-ci-eu-01	Sofia Andersen	risk_level

Q	UC	Benchmark question	Expected affected servers	Expected owners	Expected supporting fields
Q4	UC1	Is there a patch available for CVE-2026-2010? Which servers are affected?	prod-ml-us-01	Carlos Ruiz	patch_available; severity
Q5	UC1	Give me a full asset health report for SRV-006.	prod-pay-us-01	Priya Sharma	compliance_score; compliance_status; risk_level; cves_affecting
Q6	UC1	Which servers in the Frankfurt data center have unpatched vulnerabilities?	prod-web-eu-01; stg-api-eu-01	Maria Lopez; Aisha Patel	location; cves_affecting
Q7	UC2	Do you agree with the assessment of Medium risk for CVE-2026-2010 given the criticality level of prod-ml-us-01 within CMDB?	prod-ml-us-01	Carlos Ruiz	justified priority; reasoning chain; UPGRADED
Q8	UC2	Should CVE-2026-1045 on stg-api-eu-01 be escalated? The CVSS is 7.5 but it is only a staging server.	stg-api-eu-01	Aisha Patel	justified priority; reasoning chain; DOWNGRADED
Q9	UC2	Justify the risk rating of CVE-2026-1001 for the payment server SRV-006.	prod-pay-us-01	Priya Sharma	justified priority; reasoning chain; Critical
Q10	UC2	Is the Critical severity of CVE-2026-3050 appropriate for all affected servers?	prod-pay-us-01	Priya Sharma	justified priority; delta analysis result; Mission Critical
Q11	UC3	Based on our Payment Gateway exposure to CVE-2026-3050, generate a Sentinel hunt query and a summary for the patching team.	prod-pay-us-01	Priya Sharma	kql; remediation; urgency tier; P1; Spring Boot
Q12	UC3	Generate a threat hunt query for CVE-2026-1001 and provide remediation guidance for all affected servers.	prod-web-eu-01; prod-db-us-01; prod-pay-us-01	Maria Lopez; James Carter; Priya Sharma	kql; CommonSecurityLog; remediation; urgency tier; OpenSSL
Q13	UC3	CVE-2026-1045 is affecting our staging API server. Create a Sentinel query to check for SSRF exploitation and draft a patching summary.	stg-api-eu-01	Aisha Patel	kql; DnsEvents; remediation; nginx; P3
Q14	UC3	Our ML platform server is exposed to CVE-2026-2010 which has no patch. Generate a Sentinel hunt query and recommend mitigations.	prod-ml-us-01	Carlos Ruiz	kql; remediation; no patch; mitigation; network segmentation

A.3. Per-Question Scores

Table A.2 reports the weighted accuracy scores for each question under the base stochastic-noise profile. These scores were used to compute the overall benchmark means reported in Chapter 6.

Table A.2: Per-question weighted accuracy scores under the base stochastic-noise profile.

Q	UC	Parametric score	Autonomous score	Notes
Q1	UC1	77%	77%	Both approaches lost part of the expected multi-asset context under the base-noise setting; the missing components differed between paradigms.
Q2	UC1	100%	100%	Clean compliance-status lookup.
Q3	UC1	100%	65%	Autonomous output identified the affected server but missed the expected owner.
Q4	UC1	100%	100%	Single-server CVE and patch availability lookup.
Q5	UC1	100%	42%	Autonomous output missed ownership and the most expected supporting fields.
Q6	UC1	100%	82%	Autonomous output omitted one expected owner while retaining the expected field context.
Q7	UC2	100%	100%	Risk-score justification completed successfully.
Q8	UC2	100%	100%	Risk-score justification completed successfully.
Q9	UC2	100%	100%	Risk-score justification completed successfully.
Q10	UC2	100%	100%	Risk-score justification completed successfully.
Q11	UC3	100%	100%	Remediation guidance and KQL generation were completed successfully.
Q12	UC3	100%	100%	Multi-server remediation guidance was completed successfully.
Q13	UC3	100%	94%	Autonomous output omitted one expected remediation and query-support field.
Q14	UC3	100%	100%	No-patch mitigation guidance was completed successfully.
Mean	–	98.4%	90.0%	Mean weighted accuracy across all 14 benchmark questions.

A.4. Score Components

Table A.3 lists the server, owner, and field components used to compute the weighted scores in Table A.2.

Table A.3: Per-question score components under the base stochastic-noise profile.

Q	UC	Parametric			Autonomous		
		Server	Owner	Field	Server	Owner	Field
Q1	UC1	0.67	0.67	1.00	1.00	0.33	1.00
Q2	UC1	1.00	1.00	1.00	1.00	1.00	1.00
Q3	UC1	1.00	1.00	1.00	1.00	0.00	1.00
Q4	UC1	1.00	1.00	1.00	1.00	1.00	1.00
Q5	UC1	1.00	1.00	1.00	1.00	0.00	0.25
Q6	UC1	1.00	1.00	1.00	1.00	0.50	1.00
Q7	UC2	1.00	1.00	1.00	1.00	1.00	1.00
Q8	UC2	1.00	1.00	1.00	1.00	1.00	1.00
Q9	UC2	1.00	1.00	1.00	1.00	1.00	1.00
Q10	UC2	1.00	1.00	1.00	1.00	1.00	1.00
Q11	UC3	1.00	1.00	1.00	1.00	1.00	1.00
Q12	UC3	1.00	1.00	1.00	1.00	1.00	1.00
Q13	UC3	1.00	1.00	1.00	1.00	1.00	0.80
Q14	UC3	1.00	1.00	1.00	1.00	1.00	1.00

A.5. AI Disclosure

During the preparation of this thesis, artificial intelligence tools, including ChatGPT 5.5, Gemini 3.1, PaperPal, and QuillBot were used to support language editing and revision. Their use was limited to grammar checking, readability improvements, structural feedback, and refinement of academic style.

These tools were not used to generate experimental data, alter benchmark results, perform evaluations, or make independent scientific claims. The substantive content, analysis, interpretation, and final decisions remain the author's own. The author reviewed and edited all AI-assisted suggestions and takes full responsibility for the content of this thesis.