



**Università
di Genova**

DITEN DIPARTIMENTO
DI INGEGNERIA NAVALE, ELETTRICA,
ELETTRONICA E DELLE TELECOMUNICAZIONI

Master's Degree in Internet and Multimedia Engineering

**MCP-Based Adaptive Firewall Enforcement
with MITRE ATT&CK and LLM-Orchestrated
Security Automation for 5G/Cloud/Edge
Networks**

Master's Thesis

Candidate: Somaye Jafari

Supervisors: Prof. Alessandro Carrega

Prof. Roberto Bruschi

Academic Year 2025/2026

Table of Contents

Table of Contents	2
Abstract.....	8
1. Introduction	10
1.1. Background and Motivation	10
1.1.1. Scope and Role of AI in the Proposed Architecture	11
1.2. Problem Statement.....	12
1.2.1. Absence of automated integration between MITRE ATT&CK and firewall enforcement.....	12
1.2.2. Context limitations of conventional firewall policies	12
1.2.3. Response-Time Requirements for Automated 5G Security	12
1.2.4. Heterogeneous security-tool coordination	12
1.3. Research Hypothesis	13
1.4. Research Objectives	13
1.5. Research Questions	14
1.6. Contributions.....	14
1.7. Thesis Organisation	15
2. Background and Foundations	17
2.1. Overview of 5G Architecture	17
2.1.1. 5G Core (5GC).....	17
2.1.2. Service-Based Architecture (SBA)	18
2.1.3. User Plane and Control Plane Separation	19
2.2. Cloud-Native and Edge Computing in 5G	19
2.2.1. Network Functions Virtualisation (NFV) and Cloud-Native Network Functions (CNFs)	19
2.2.2. Kubernetes and Service Mesh.....	20
2.2.3. Multi-access Edge Computing (MEC).....	20
2.3. Firewall Architectures and Their Limitations	21
2.3.1. Packet Filtering and Stateful Firewalls	21
2.3.2. Next-Generation Firewalls (NGFW)	22
2.3.3. Limitations of Firewall Architectures in 5G Environments	22
2.4. Threat Landscape in 5G Networks	23
2.4.1. Control Plane Attacks (SBA/API Abuse).....	24
2.4.2. User Plane Attacks (DDoS and GTP-U Abuse)	25

2.4.3. Network Slice-Oriented Threats.....	26
2.4.4. Cloud and Edge Infrastructure Threats	26
2.5. Model Context Protocol (MCP)	27
2.5.1. Architecture and Components.....	28
2.5.2. Tool Invocation Model.....	29
2.5.3. Security Considerations for MCP.....	29
2.6. MITRE ATT&CK Framework.....	30
2.6.1. Structure and Organisation	30
2.6.2. STIX 2.0 Representation	31
2.6.3. Techniques Relevant to 5G Environments	31
2.7. Chapter Summary	32
3. Related Work	33
3.1. Machine Learning and Deep Learning-Based Intrusion Detection.....	33
3.2. Adaptive, Programmable, and Automated Security Enforcement.....	34
3.2.1. SDN and eBPF-Based Enforcement	34
3.2.2. Intent-Based and Automatic Policy Refinement	35
3.2.3. Security Orchestration, Automation, and Response (SOAR)	35
3.3. LLM-Based Security Automation	36
3.4. MCP-Based Systems and MCP Security	37
3.5. Machine Learning for 5G and Network Slice Security	38
3.6. Federated Learning in Network Security	38
3.7. Gap Analysis.....	39
3.8. Chapter Summary	41
4. System Architecture	42
4.1. Architecture Overview.....	42
4.2. Detection and Feature Extraction Module	43
4.3. MITRE ATT&CK MCP Server.....	45
4.4. Firewall Enforcement MCP Server	45
4.5. LLM Orchestrator.....	46
4.6. End-to-End Detection-to-Enforcement Pipeline.....	47
4.7. Chapter Summary	49
5. Implementation.....	50
5.1. Development Environment.....	50
5.2. MITRE ATT&CK MCP Server Implementation.....	51

5.3. Firewall Enforcement MCP Server Implementation	52
5.4. Evaluation Pipeline Implementation.....	53
5.5. End-to-End Workflow Verification	54
5.6. Chapter Summary	54
6. Experimental Methodology	55
6.1. Testbed Configuration	55
6.2. Synthetic Dataset	56
6.3. Detection Rule Derivation	57
6.4. Performance Metrics and Latency Measurement	58
6.4.1. Detection Performance Metrics	58
6.4.2. Static Packet-Count Threshold Baseline.....	58
6.4.3. Pipeline Latency Measurement.....	59
6.5. Sequential Input-Set Scaling Experiment Design	59
6.6. Repeated Execution and Statistical Considerations	60
7. Results and Discussion.....	62
7.1. Detection Performance	62
7.1.1. Cross-Dataset Validation	62
7.1.2. Robustness Under Feature Perturbation	64
7.2. Threats to Validity	67
7.2.1. Internal Validity	68
7.2.2. Construct Validity	68
7.2.3. External Validity	70
7.2.4. Conclusion Validity	70
7.3. Sequential Input-Set Scaling with Attacker-Set Size	71
7.3.1. Discussion of Detection Scaling Results.....	72
7.4. Pipeline Latency Analysis	73
7.4.1. Latency Decomposition and Bottleneck Identification.....	74
7.5. Failure Cases and Error Analysis	76
7.5.1. Development-Time Failures.....	76
7.5.2. Evaluation-Time Failures	76
7.6. LLM-Based Detection Without Rule Filtering.....	77
7.6.1. Baseline LLM	80
7.6.2. Prompt Engineering	81
7.6.3. Relational Features	81

7.6.4. Two-Branch Prompt (Negative Result)	82
7.6.5. Repeatability	84
7.6.6. Infrastructure Reliability	85
7.6.7. Discussion.....	85
7.7. LLM-Based Detection With MITRE MCP Retrieval (Stage 3)	87
7.7.1. Experiment 3A — Mechanism Validation	88
7.7.2. Experiments 3B / 3F — Does the Model Voluntarily Use MCP?	88
7.7.3. Experiment 3C — Classification Without Any MITRE Information	89
7.7.4. Experiments 3D vs. 3E — Does Retrieved MITRE Knowledge Help?.....	89
7.7.5. Experiment 3G — Forced Agentic Tool Use	90
7.7.6. Experiment 3H — Isolating Prompt Length From Semantic Content.....	91
7.7.7. Summary of Research Questions Answered	92
7.7.8. Discussion.....	93
7.8. Summary of Findings.....	94
8. Security Analysis	97
8.1. Threat Model.....	97
8.2. Prompt Injection via MITRE Technique Descriptions	97
8.3. Tool Abuse: Blocking Legitimate IP Addresses	98
8.4. Privilege Escalation and Command Injection at the Enforcement Boundary	98
8.5. LLM Hallucination: Invalid or Fabricated Actions	100
8.6. Compromised or Malicious MCP Server: Mitigation Injection	102
8.7. Human-in-the-Loop as a Compensating Control	102
8.8. Summary of Identified Risks and Mitigations.....	103
9. Conclusion and Future Work.....	105
9.1. Summary of Contributions	105
9.2. Answers to the Research Questions	106
9.3. Limitations.....	108
9.4. Architectural Applicability to 5G Deployments	110
9.5. Lessons Learned.....	111
9.6. Future Work.....	112
9.6.1. Immediate Priorities	112
9.6.2. Medium-Term Priorities	113
9.6.3. Long-Term Priorities.....	113
9.6.4. Additional Directions	114

References	115
Appendix A: Configuration Files and Parameters	119
A.1. Ollama Orchestrator Configuration	119
A.2. MITRE ATT&CK MCP Server (<code>mcp_server.py</code>)	119
A.3. Firewall Enforcement MCP Server (<code>router_firewall.py</code>)	119
A.4. Evaluation Pipeline (<code>evaluate_mcp_pipeline.py</code>)	120
A.5. Supplementary Evaluation Scripts	120
A.6. MCP Servers Configuration File (<code>mcp_router_config.json</code>)	120
Appendix B: Additional Results	122
B.1. Full Scalability Results (All Techniques, $n = 1-10$)	122
B.2. Full Perturbation Experiment Results ($N = 100$ Trials)	123
B.3. Latency Reproducibility Diagnostic (Section 7.4)	124
Appendix C: Source Code Overview	125
C.1. <code>mcp_server.py</code> (MITRE ATT&CK MCP Server, Section 5.2)	125
C.2. <code>router_firewall.py</code> (Firewall Enforcement MCP Server, Section 5.3)	125
C.3. <code>evaluate_mcp_pipeline.py</code> (Evaluation Pipeline, Section 5.4)	125
C.4. Supplementary Rule-Based Evaluation Scripts (Sections 7.1.1, 7.1.2)	126
C.5. LLM-Based Detection Scripts (Sections 7.6, 7.7)	126
C.6. Patch Guidance: T1495 $\rightarrow$ T1498 Identifier Correction for Stage 3 MCP Queries.....	127

Statement on the Use of Artificial Intelligence

An AI assistant (Claude Sonnet 5, developed by Anthropic) was used during the preparation of this thesis, specifically for writing and editing support: polishing academic prose, identifying inconsistencies across chapters, locating and verifying bibliographic references, and assisting with document formatting. All research design, system implementation, experimental execution, and scientific and methodological decisions presented in this thesis were carried out independently by the author.

Abstract

Fifth-generation (5G) mobile networks introduce a service-based, cloud-native, and network-sliced architecture that expands the security attack surface beyond what conventional, rule-centric firewall enforcement was designed to address. Meanwhile, the MITRE ATT&CK framework provides a structured knowledge base of adversary techniques and mitigations, but translating this natural-language guidance into executable enforcement actions typically requires manual analyst interpretation. This thesis investigates an architecture in which the Model Context Protocol (MCP) coordinates a deterministic traffic-classification component, a MITRE ATT&CK knowledge-retrieval component, and a large language model (LLM) orchestrator to automate the translation from a detected attack technique to a live firewall enforcement action.

A proof-of-concept system is designed and implemented, comprising two MCP servers — one exposing MITRE ATT&CK Enterprise data, the other exposing iptables-based firewall enforcement on a single VPN-connected router — coordinated by an LLM orchestrator (via ollmcp/Ollama). The system is evaluated against synthetic traffic representing five MITRE ATT&CK techniques (T1190, T1048, T1059, T1498, and T1568), using a dataset-calibrated, rule-based detection engine as the primary classification mechanism, with the LLM/MCP stage handling threat interpretation and mitigation orchestration downstream of detection.

The detection engine achieves perfect classification metrics (Precision = Recall = F1 = 1.000) on the evaluated synthetic dataset, substantially exceeding a technique-agnostic static packet-count threshold baseline; however, this result reflects calibration to the specific dataset used for both rule derivation and testing, and a feature-perturbation stress test shows that three of the five techniques degrade markedly in Recall once traffic features drift 10–20% or more from their calibration values, while two techniques retain robustness for distinct reasons examined in detail. The end-to-end detection-to-enforcement pipeline exhibits a representative latency of 1.36–2.32 seconds, dominated by the MCP/LLM stage rather than by detection or SSH-based enforcement — a latency three orders of magnitude above the sub-millisecond budgets associated with Ultra-Reliable Low-Latency Communication (URLLC) traffic classes, indicating that the current architecture is unsuitable for direct in-line mitigation of latency-critical 5G traffic without further architectural changes.

A complementary set of experiments investigates whether an LLM can perform this detection task directly, without the deterministic rule layer, and whether MITRE ATT&CK knowledge accessed via MCP improves such detection. Direct LLM-based classification, even after substantial prompt engineering and the introduction of relational population-level features, does not match the deterministic detector’s performance and fails to detect two of five techniques in its best configuration; a ten-run repeatability check yields a mean Micro-F1 of 0.893 (± 0.002). Across eight further controlled experiments, MCP tool availability alone does not improve LLM classification, the LLM does not necessarily invoke an available retrieval tool voluntarily, rich contextual traffic features alone achieve near-perfect classification without any MITRE information, and official qualitative MITRE ATT&CK descriptions — whether provided deterministically or retrieved agentially

through MCP — consistently underperform engineered quantitative flow-level descriptions, a degradation that persists after controlling for prompt length.

These findings collectively support an architectural conclusion: deterministic, low-latency detection should remain upstream of MCP-mediated LLM reasoning, with the latter reserved for threat interpretation, MITRE ATT&CK retrieval, and mitigation orchestration rather than for the detection task itself. This thesis is a proof-of-concept evaluated against synthetic traffic and a single-router enforcement testbed; no 5G Core, Service-Based Architecture traffic, PFCP, GTP-U, or User Plane Function is evaluated, and the architectural applicability of the proposed pattern to 5G Core, Multi-access Edge Computing, and cloud-native deployments is analysed separately from, and should not be inferred directly from, the empirical results presented. A security analysis further identifies command-injection and privilege-escalation risks at the enforcement boundary, prompt-injection and tool-poisoning risks in the MCP tool chain, and corresponding mitigations, positioning the security of the MCP-mediated reasoning and enforcement pathway as an integral part of the system's threat model rather than an afterthought.

1. Introduction

1.1. Background and Motivation

Fifth-generation (5G) mobile networks introduce substantial architectural and operational changes relative to previous mobile-network generations. Key performance indicators defined by the International Telecommunication Union (ITU) for 5G include peak data rates of up to 20 Gbps, user-plane latency targets as low as 1 millisecond for selected Ultra-Reliable Low-Latency Communication (URLLC) scenarios, and connection densities of up to one million devices per square kilometre for massive Machine-Type Communication (mMTC) [5]. These characteristics have motivated 5G use cases in areas such as industrial automation, connected transportation, immersive applications, and latency-sensitive remote interaction.

The architectural foundations of 5G differ substantially from those of previous generations. The 5G Core (5GC) adopts a Service-Based Architecture (SBA) in which network functions expose capabilities as HTTP/2-based APIs, replacing the point-to-point interface model of 4G [6]. Network Functions Virtualisation (NFV) enables the deployment of network functions as software on commodity hardware, while network slicing allows multiple logically isolated virtual networks to coexist on shared physical infrastructure [9]. Multi-access Edge Computing (MEC) further distributes processing resources to the network periphery, reducing latency for geographically proximate users.

While these architectural innovations yield substantial operational benefits, they simultaneously expand the attack surface available to adversaries. The proliferation of HTTP/2 API endpoints across the SBA introduces vulnerabilities to API abuse, unauthorised access, and injection attacks [10]. The virtualised, containerised deployment of network functions exposes the infrastructure to container escape, supply chain compromise, and orchestration platform exploitation [11]. Network slicing introduces inter-slice isolation requirements whose failure can enable lateral movement between logical network domains [34]. These threat vectors are substantially different in character from those addressed by conventional network security mechanisms.

Traditional firewall architectures — encompassing stateless packet filters, stateful inspection firewalls, and Next-Generation Firewalls (NGFWs) — were designed for centralised, relatively static network environments. Traditional rule-centric firewall deployments commonly depend on manually authored or administratively managed policies, which can become difficult to maintain in highly dynamic environments characterised by frequent workload instantiation, migration, and scaling [12]. Furthermore, conventional IP/port-centric policy models primarily operate on network- and transport-layer attributes such as source address, destination address, protocol, and port number, and therefore require additional integration to incorporate 5G-specific context such as network slice identifiers, subscriber identities, and service types.

The MITRE ATT&CK framework provides a structured, continuously maintained knowledge base of adversary tactics, techniques, and procedures (TTPs) observed in real-world attacks [7]. For each documented technique, the framework provides descriptions of

adversary behaviour, detection guidance, and recommended mitigations. MITRE ATT&CK mitigation guidance is expressed primarily as human-readable defensive recommendations rather than directly executable network policies. Translating this guidance into concrete enforcement actions therefore requires an additional interpretation and policy-generation layer, which is commonly implemented through analyst-driven procedures or application-specific automation [14].

Large Language Models (LLMs) have demonstrated significant capability in natural language understanding, code generation, and structured reasoning [13]. Their potential application to the automated interpretation of threat intelligence and selection of mitigation actions has attracted considerable research interest. However, the effective integration of LLMs into operational security workflows requires a structured mechanism for providing access to relevant tools and data sources. The Model Context Protocol (MCP), developed by Anthropic, addresses this requirement by defining an open protocol through which LLM agents can invoke external tools, query knowledge bases, and execute actions in a controlled manner [8].

The primary motivation for this research is to investigate an integrated architecture connecting three domains that are typically treated separately in the literature reviewed in Chapter 3: structured MITRE ATT&CK threat intelligence, LLM-based reasoning and tool orchestration through MCP, and dynamic firewall enforcement. In this thesis, the term *adaptive* refers to dynamic mitigation selection based on the detected ATT&CK technique and retrieved mitigation context, rather than reliance on a fixed, technique-agnostic blocking rule. It does not imply online learning, continual adaptation of detection models, or autonomous model retraining; all models and detection rules remain fixed during evaluation.

1.1.1. Scope and Role of AI in the Proposed Architecture

The implemented and evaluated system is a proof-of-concept. Evaluation uses synthetic packet captures representing five MITRE ATT&CK techniques, while enforcement is applied through iptables on a single router reachable over an OpenVPN tunnel. The testbed does not contain a 5G Core implementation, SBA traffic, PFCP, GTP-U, a UPF, or cloud-native CNFs. Consequently, empirical performance claims are limited to the environment described in Chapter 6; references to 5G applicability denote architectural applicability unless an empirical result is explicitly stated. Extensions required for deployment in 5G Core, MEC, network slicing, and cloud-native environments are analysed in Section 9.4, and no detection, latency, or enforcement result presented in Chapter 7 should be interpreted as direct evidence of performance in a production 5G Core network.

Traffic classification in the proof-of-concept is deterministic and rule-based rather than learned or LLM-based, while the LLM operates downstream to support MITRE ATT&CK retrieval, threat interpretation, and mitigation selection and orchestration through MCP. This separation is a deliberate architectural choice supported by the experiments in Sections 7.6 and 7.7, where direct LLM-based classification achieves lower performance than the dataset-calibrated deterministic detector under the evaluated conditions. Because the architecture permits an LLM-mediated agent to invoke live enforcement actions, the

security of the MCP tool boundary is itself part of the system’s threat model and is analysed separately in Chapter 8.

1.2. Problem Statement

Despite recent advances in automated and AI-assisted network security, several important challenges remain unaddressed in the context of 5G network protection.

1.2.1. Absence of automated integration between MITRE ATT&CK and firewall enforcement

The MITRE ATT&CK framework documents mitigations for each catalogued technique in natural language. The literature reviewed for this thesis did not identify a standardised, automated end-to-end mechanism to translate these mitigation descriptions into executable firewall rules. In many workflows, security analysts or application-specific automation must interpret MITRE recommendations and translate them into concrete enforcement policies, a process that can be time-consuming, brittle, and difficult to generalise across tools [14]. The proof-of-concept presented in this thesis addresses a constrained instance of this general problem: it operationalises a bounded subset of network-relevant mitigations via predefined firewall actions selected by an LLM orchestrator, rather than performing open-ended translation of arbitrary mitigation text into arbitrary firewall syntax (Section 4.4).

1.2.2. Context limitations of conventional firewall policies

Conventional IP/port-based firewall rules primarily evaluate network- and transport-layer attributes such as source address, destination address, protocol, and port. Without integration with 5G control-plane or orchestration context, such rules cannot directly incorporate attributes such as network slice identifiers, subscriber identities, or service-specific state [15].

1.2.3. Response-Time Requirements for Automated 5G Security

Fast-moving threats such as volumetric denial-of-service and user-plane flooding can degrade network resources rapidly. Security automation intended for latency-sensitive network environments must therefore minimise the time between detection and enforcement. Manual policy modification may be inadequate for such scenarios, motivating automated response mechanisms. This motivates the need to evaluate the response-time cost introduced by an LLM-mediated response architecture and assess its compatibility with representative documented 5G latency targets, an issue examined experimentally in Section 7.4 [16].

1.2.4. Heterogeneous security-tool coordination

Security platforms commonly expose heterogeneous APIs, data models, and tool interfaces, requiring application-specific integration when combining detection, threat intelligence, reasoning, and enforcement components. The lack of a uniform agent-oriented tool interface complicates the construction of reusable LLM-mediated security workflows [17].

The literature reviewed in Chapter 3 did not identify an end-to-end architecture that combines MITRE ATT&CK threat intelligence, MCP-mediated LLM reasoning, and live firewall enforcement within a single experimentally evaluated workflow, while explicitly examining both the benefits and limitations of placing LLM reasoning in the security-response pipeline.

1.3. Research Hypothesis

Based on the problem statement above, this thesis investigates the following hypothesis:

A technique-aware detection-to-enforcement architecture combining deterministic traffic classification with MCP-mediated MITRE ATT&CK retrieval and LLM-assisted mitigation orchestration is expected to provide more accurate technique-specific classifications than the evaluated technique-agnostic packet-count threshold baseline, while introducing additional latency and MCP/LLM-stage operational-reliability costs.

The hypothesis is evaluated in Chapter 7. RQ3 assesses whether the deterministic technique-specific classifier improves upon the evaluated static threshold baseline, while RQ4 characterises the latency and operational costs introduced by the MCP/LLM stage. Complementary experiments in Sections 7.6 and 7.7 investigate the viability of direct LLM-based and MITRE/MCP-assisted classification as alternatives to the deterministic detection stage. The resulting evidence motivates the final separation between deterministic detection and MCP-mediated reasoning and enforcement.

Detection quality is evaluated using Precision, Recall, and F1, whereas enforcement feasibility is evaluated through functional tool execution, latency, and MCP/LLM-stage operational reliability; these metrics assess different components of the architecture and are interpreted separately throughout Chapter 7.

1.4. Research Objectives

This thesis pursues the following five objectives:

Objective 1: Design and implement a MITRE ATT&CK MCP server. Develop an MCP server that exposes the MITRE ATT&CK knowledge base to LLM agents through standardised tool interfaces, enabling programmatic retrieval of technique descriptions and associated mitigations by technique identifier.

Objective 2: Design and implement a firewall enforcement MCP server. Develop an MCP server that accepts structured firewall action requests and applies them through iptables on a networked enforcement node accessible via SSH over an OpenVPN tunnel. The server shall maintain a record of active blocking rules and expose status query functionality.

Objective 3: Construct an automated detection-to-enforcement pipeline. Integrate the MITRE MCP server and firewall MCP server with an LLM orchestrator to form a complete pipeline from attack detection to enforcement. Conceptually, the pipeline supports the following sequence: detect attack technique → retrieve relevant MITRE ATT&CK information → select an appropriate mitigation action → invoke the firewall MCP

enforcement tool → apply the corresponding predefined iptables action. The principal deterministic evaluation (Chapter 7) exercises a constrained variant of this sequence, as detailed in Section 4.6.

Objective 4: Evaluate system performance across multiple attack scenarios. Evaluate the proposed system against synthetic traffic representative of five MITRE ATT&CK techniques (T1190, T1048, T1059, T1498, T1568) using precision, recall, F1 score, and pipeline latency as performance metrics. Assess detection scaling by varying the attacker-set size from one to ten (Section 7.3), evaluating computational scaling with input size rather than true runtime concurrency.

Objective 5: Analyse applicability to 5G network environments. Provide an architectural analysis of how the proposed proof-of-concept system can be extended to 5G Core deployments, addressing considerations of network slicing, MEC integration, NFV compatibility, and 5G-specific protocol support.

1.5. Research Questions

This thesis addresses the following research questions:

ID	Research Question	Evaluation Approach
RQ1	Can an MCP server reliably retrieve MITRE ATT&CK technique data and present it in a form suitable for LLM consumption?	Functional testing of MITRE MCP server tools
RQ2	How can MITRE ATT&CK mitigation guidance be operationalised through an LLM orchestrator and MCP-based firewall enforcement interface?	Architecture implementation, tool-invocation validation, end-to-end evaluation
RQ3	Does the technique-specific detection component used by the proposed architecture achieve higher Precision, Recall, and F1 than the evaluated static threshold baseline on the synthetic dataset?	Synthetic-dataset comparison with 30 deterministic consistency-verification executions
RQ4	What is the end-to-end latency of the detection-to-enforcement pipeline, and how does it compare with representative documented 5G latency targets (e.g., for eMBB and URLLC scenarios)?	Latency measurement vs. documented 5G latency targets
RQ5	How does detection performance and per-sample classification latency scale as the attacker-set size increases from one to ten?	Input-set scaling experiment (not runtime concurrency)

1.6. Contributions

This thesis makes the following contributions:

1. **An MCP-based architecture for automated adaptive firewall management.** This thesis proposes an architecture in which MCP serves as the coordination layer between an LLM orchestrator, a threat intelligence source (MITRE ATT&CK), and a network enforcement engine (iptables). The separation introduced by the MCP

interface is intended to permit alternative enforcement backends to replace the iptables proof-of-concept without modifying the higher-level reasoning architecture.

2. **A MITRE ATT&CK MCP server implementation.** An MCP server is implemented that exposes MITRE ATT&CK Enterprise data through tool interfaces compatible with MCP-capable LLM agents. The server supports querying by technique identifier and returns structured descriptions, mitigation guidance filtered by network-relevance, and reference URLs.
3. **A firewall enforcement MCP server with iptables integration.** An MCP server is implemented that translates structured firewall action requests into iptables commands applied on a remote enforcement node via SSH. The server provides blocking, unblocking, status query, and statistics functions accessible through the MCP tool interface.
4. **An empirical evaluation framework for the proposed detection-to-enforcement architecture.** A complete evaluation framework is developed and applied, encompassing synthetic traffic generation representative of five MITRE ATT&CK techniques, a rule-based detection engine with documented feature derivation, 30-execution consistency verification with explicit discussion of its statistical interpretation, detection scaling testing across attacker-set sizes from one to ten, and comparison against a technique-agnostic static packet-count threshold baseline.
5. **An architectural analysis of 5G deployment considerations.** A structured analysis is provided of the extensions required to deploy the proposed architecture in 5G Core environments, addressing network slicing, UPF integration, MEC deployment, and 5G-specific protocol support, together with an explicit assessment of current limitations.
6. **A systematic empirical investigation of direct LLM-based traffic classification and MITRE/MCP-assisted classification.** The thesis evaluates direct LLM classification without the deterministic rule layer, investigates the effects of prompt structure, relational feature engineering, task decomposition, and repeatability, and conducts controlled experiments on MITRE ATT&CK retrieval through MCP. Under the evaluated conditions, the results show the importance of quantitative and relational traffic context, the limited effectiveness of qualitative ATT&CK descriptions as a substitute for engineered quantitative flow-level detection knowledge, and the rationale for retaining deterministic classification upstream of MCP-mediated reasoning and mitigation selection.

1.7. Thesis Organisation

The remainder of this thesis is organised as follows.

Chapter 2: Background and Foundations provides the technical background necessary to contextualise the proposed system, covering the 5G Core architecture, cloud-native and

edge deployment, firewall architectures, the 5G threat landscape, the MCP protocol, and the MITRE ATT&CK framework.

Chapter 3: Related Work reviews prior research in ML-based intrusion detection, adaptive firewall architectures, LLM-based security automation, and MCP security, and identifies the specific gap that motivates the present work.

Chapter 4: System Architecture presents the proposed architecture, including the detection module, the two MCP servers, the LLM orchestrator integration, and the complete detection-to-enforcement pipeline.

Chapter 5: Implementation describes the development environment, implementation details of each system component, and the end-to-end workflow verification procedure.

Chapter 6: Experimental Methodology defines the evaluation framework, including the testbed, synthetic dataset, detection rule derivation, performance metrics, and experimental design for latency, input-set scaling, and repeated executions.

Chapter 7: Results and Discussion presents the experimental evaluation of the deterministic detection-to-enforcement pipeline, including detection performance, robustness, scaling, latency, and failure analysis. It also reports complementary experiments on direct LLM-based detection and MITRE/MCP-assisted classification, and synthesises their implications for the final architecture.

Chapter 8: Security Analysis examines the security properties of the proposed system itself, including MCP-specific threats such as prompt injection, tool abuse, and LLM hallucination, together with corresponding mitigations.

Chapter 9: Conclusion and Future Work summarises the contributions, answers the research questions, discusses limitations, analyses architectural applicability to 5G deployments, and identifies directions for future work.

2. Background and Foundations

2.1. Overview of 5G Architecture

Compared with earlier mobile-network generations, 5G places substantially greater emphasis on service-based interfaces, control/user-plane separation, virtualised and cloud-native network functions, and distributed deployment. These architectural choices improve flexibility and scalability while also introducing additional security considerations [6,9].

Figure 2.1 provides a reference overview of the 5G System, showing the User Equipment (UE), the NG-RAN, the principal network functions of the Service-Based Architecture (SBA), and the user-plane path through the UPF to the data network, including the N2, N3, N4, N6, and N11 reference points discussed in the following subsections.

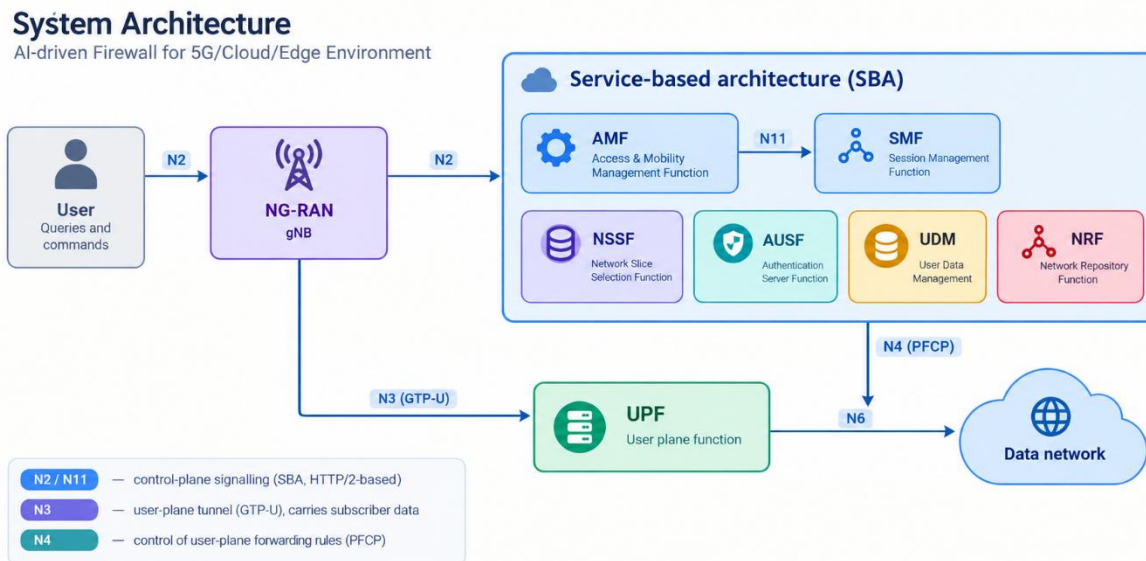


Figure 2.1: Reference overview of the 5G System architecture. The UE connects via the NG-RAN (gNB) to the Service-Based Architecture, which comprises the AMF, SMF, NSSF, AUSF, UDM, and NRF connected via service-based interfaces (e.g., N11). Control-plane signalling reaches the AMF via N2. The user plane is forwarded from the NG-RAN to the UPF via N3 (GTP-U) and onward to the data network via N6, while the SMF controls UPF forwarding behaviour via N4 (PCFP). *Source: Author's own diagram, based on 3GPP TS 23.501 [6].*

2.1.1. 5G Core (5GC)

The 5G Core (5GC) constitutes the central control and management layer of the 5G network, responsible for user authentication, mobility management, session establishment, subscriber data handling, and interconnection with external data networks. In contrast to the 4G Evolved Packet Core (EPC), the 5GC is designed around software-based network

functions and supports virtualised and cloud-native deployment models on general-purpose computing infrastructure [6].

The principal network functions (NFs) comprising the 5GC are as follows:

Access and Mobility Management Function (AMF): The AMF serves as the primary point of contact for user equipment (UE) during network registration. It is responsible for access control, mobility tracking, handover coordination, and interfacing with authentication functions to verify UE identity [6].

Session Management Function (SMF): The SMF manages the lifecycle of user sessions, including IP address allocation, session policy and QoS control, and selection and configuration of the appropriate User Plane Function (UPF). Upon session termination, the SMF is responsible for releasing all associated resources [6].

User Plane Function (UPF): The UPF serves as the data forwarding entity within the 5GC, handling packet forwarding, policy enforcement, and usage reporting for charging purposes. As the gateway between the core network and external data networks, the UPF is a critical component from both a performance and security perspective [6].

Unified Data Management (UDM): The UDM maintains subscriber profiles, including subscription data, authentication credentials, and service entitlements. It acts as the authoritative repository for user-related information within the 5GC [6].

Authentication Server Function (AUSF): The AUSF performs mutual authentication between the network and the UE, implementing the 5G-AKA and EAP-AKA' authentication protocols as defined by 3GPP [6].

NF Repository Function (NRF): The NRF provides service discovery capabilities within the 5GC, maintaining a registry of available network functions and their endpoints. NFs consult the NRF to locate other functions prior to establishing communication [6].

Network Slice Selection Function (NSSF): The NSSF determines the appropriate network slice for each UE based on subscription data and operator policies, enabling the coexistence of multiple logical networks on shared physical infrastructure [6].

2.1.2. Service-Based Architecture (SBA)

A defining characteristic of the 5GC is its adoption of a Service-Based Architecture (SBA), in which network functions expose their capabilities as services through standardised HTTP/2-based APIs [6]. This paradigm replaces the point-to-point interface model used in previous generations, where each function maintained dedicated bilateral connections with every other function it interacted with.

Under SBA, network functions operate as both service producers and consumers. A network function offering a service registers its NF profile and available services with the NRF. A consumer NF can subsequently query the NRF to discover an appropriate service producer and invoke the selected service through the corresponding service-based interface. This decoupled model reduces inter-function dependencies and facilitates

independent scaling and deployment of individual NFs, drawing parallels to microservice architectures widely adopted in cloud-native software engineering [18].

From a security standpoint, SBA introduces a substantially enlarged API attack surface. Each exposed service endpoint represents a potential target for unauthorised access, parameter manipulation, injection attacks, or denial-of-service attempts. 3GPP defines security mechanisms including TLS-based protection and OAuth 2.0-based service authorisation for applicable SBA interactions; however, the complexity of the API ecosystem makes comprehensive enforcement challenging [19].

2.1.3. User Plane and Control Plane Separation

A foundational architectural principle of the 5GC is the separation of control-plane functions — encompassing signalling and management functions such as authentication, session management, and mobility handling — from the UPF-based user plane, which is dedicated to the forwarding of end-user data traffic [6].

This separation confers several operational advantages. User Plane Functions can be deployed independently of control plane entities, enabling distributed placement at the network edge to reduce latency for delay-sensitive applications. Furthermore, the user plane can be scaled horizontally in response to traffic demand without necessitating changes to control plane components [6].

The interface between the SMF and UPF, designated N4, utilises the Packet Forwarding Control Protocol (PFCP) to convey forwarding rules and session management commands from the control plane to the user plane. The N3 interface, connecting the base station (gNB) to the UPF, employs the GTP-U (GPRS Tunnelling Protocol – User Plane) protocol for encapsulation of user data [20], [21].

From a security perspective, the N4 and N3 interfaces represent significant attack vectors. Adversaries may attempt to exploit PFCP message manipulation to alter packet forwarding behaviour, or to abuse GTP-U tunnelling mechanisms through traffic injection, address spoofing, or Tunnel Endpoint Identifier (TEID) hijacking [23]. These threats are examined in further detail in Section 2.4.

2.2. Cloud-Native and Edge Computing in 5G

The deployment of 5G networks is intrinsically coupled with two foundational paradigms: cloud-native computing and edge computing. These paradigms enable the flexibility, scalability, and low-latency operation that 5G service requirements demand, while simultaneously introducing new operational and security challenges.

2.2.1. Network Functions Virtualisation (NFV) and Cloud-Native Network Functions (CNFs)

Network Functions Virtualisation (NFV) is the practice of decoupling network functions from proprietary hardware appliances and implementing them as software running on commodity servers [9]. In traditional network deployments, functions such as firewalls, load balancers, and session gateways were realised as dedicated physical devices. NFV

replaces these with software-based equivalents, referred to as Virtual Network Functions (VNFs), which can be instantiated, scaled, and terminated on demand.

Network-function deployment has increasingly expanded from VM-based VNFs toward containerised CNFs, particularly in cloud-native environments; VNFs and CNFs nevertheless coexist in contemporary telecommunications infrastructures [24]. Containers generally incur lower startup and resource overhead than full virtual machines, although actual instantiation time depends on the workload, orchestration platform, and deployment environment.

From a security perspective, the shift toward CNFs introduces several considerations. Container images sourced from public registries may contain vulnerabilities or malicious components if not rigorously validated [11]. Because containers commonly share the host kernel, a container escape may expose co-resident workloads through a shared host boundary; VM-based isolation introduces a different, generally stronger hypervisor boundary, although it is not immune to escape or host compromise.

2.2.2. Kubernetes and Service Mesh

The orchestration of CNFs at scale requires a robust container management platform. Kubernetes has emerged as the de facto standard for this purpose, providing automated deployment, scaling, self-healing, and service discovery for containerised workloads [25]. In the context of 5G, Kubernetes serves as the infrastructure layer upon which CNFs such as AMF, SMF, and UPF are deployed and managed.

Kubernetes provides several security-relevant capabilities, including Role-Based Access Control (RBAC) for limiting access to cluster resources, network policies for controlling inter-pod traffic, and secrets management for protecting sensitive configuration data. However, misconfigurations of these mechanisms represent a significant source of vulnerability. Studies have identified exposed Kubernetes API servers, overly permissive RBAC policies, and unencrypted etcd stores as common weaknesses in production deployments [26].

Complementing Kubernetes, a service mesh provides a dedicated infrastructure layer for managing inter-service communication. Many service-mesh deployments use sidecar proxies associated with individual workloads, although newer architectures may employ alternative dataplane models [27]. This architecture enables mutual TLS (mTLS) encryption between all services, fine-grained authorisation policies, distributed tracing, and traffic rate limiting — capabilities that are directly relevant to securing the SBA communication fabric of the 5GC.

2.2.3. Multi-access Edge Computing (MEC)

Multi-access Edge Computing (MEC), standardised by ETSI, refers to the deployment of computing and storage resources at the edge of the radio access network, in close physical proximity to end users [28]. By placing compute resources closer to users, MEC can reduce transport latency and support latency-sensitive applications such as industrial automation, vehicle-to-everything (V2X) communication, and remote surgical assistance, although

achieved end-to-end latency depends on radio, transport, processing, and application components.

In 5G deployments, an edge-positioned UPF can support local breakout toward nearby MEC applications, reducing the need to route user traffic through a distant central data network. The SMF configures these edge UPFs via the N4 interface, directing traffic matching specific criteria to local application servers [6].

MEC introduces a distinct set of security challenges relative to centralised cloud deployments. Edge nodes may be more resource-constrained than central cloud facilities, potentially limiting the computational overhead available for security functions such as deep packet inspection or machine learning-based anomaly detection [28]. The geographically distributed nature of edge deployments also complicates centralised security monitoring and policy enforcement. Furthermore, edge nodes may be situated in physically insecure locations, increasing the risk of hardware tampering and unauthorised physical access. Application mobility, whereby workloads migrate between edge nodes in response to user movement, necessitates dynamic security policy propagation to ensure consistent enforcement across locations [28].

2.3. Firewall Architectures and Their Limitations

Firewalls are a core mechanism for enforcing network security policy at traffic-control boundaries. Over several decades of development, firewall technology has evolved through multiple generations, each addressing limitations of its predecessor. This section reviews the principal architectural paradigms and examines their limitations in the context of 5G network environments.

2.3.1. Packet Filtering and Stateful Firewalls

The earliest generation of firewalls operated on the principle of stateless packet filtering. Each packet is evaluated independently against a predefined ruleset based on a fixed set of header fields: source IP address, destination IP address, source port, destination port, and transport protocol [29]. Rules are applied sequentially, and the first matching rule determines whether the packet is permitted or dropped. This approach is computationally efficient and can be implemented at line rate in hardware, but it offers no awareness of application-layer content or the relational context between packets belonging to the same communication session.

The fundamental limitation of stateless packet filtering — its inability to distinguish between legitimate response traffic and unsolicited inbound packets — motivated the development of stateful inspection firewalls [29]. A stateful firewall maintains a connection tracking table, recording the state of active TCP, UDP, and ICMP flows. When a packet arrives, the firewall consults this table to determine whether it corresponds to an established or related connection, permitting response traffic without requiring explicit inbound rules. This mechanism significantly reduces the permissiveness of rulesets while preserving correct operation for connection-oriented protocols.

Classic stateless packet filtering and stateful inspection primarily operate at Layers 3 and 4 (OSI network and transport layers). They are typically incapable of inspecting application-layer payloads, identifying application-specific protocols regardless of port number, or detecting threats embedded within otherwise conformant traffic flows [29].

2.3.2. Next-Generation Firewalls (NGFW)

Next-Generation Firewalls (NGFWs) extend the capabilities of stateful inspection by integrating application-layer visibility and additional security functions into a unified platform [30]. The defining capabilities of NGFWs include the following:

Deep Packet Inspection (DPI): NGFWs inspect the full payload of network packets, enabling identification of application protocols irrespective of port number, detection of protocol anomalies, and extraction of application-layer attributes for policy enforcement [31].

Intrusion Detection and Prevention (IDS/IPS): NGFWs incorporate signature-based and, in some implementations, anomaly-based detection engines capable of identifying known attack patterns and blocking malicious traffic in real time [31].

Application Identification: Through behavioural analysis and protocol fingerprinting, NGFWs can identify specific applications and apply differentiated policies based on application identity rather than port and protocol alone [31].

Identity-Based Policy Enforcement: NGFWs can integrate with directory services such as LDAP or Active Directory to associate network traffic with authenticated user identities, enabling user-aware access control policies [31].

TLS Inspection: Given the widespread adoption of TLS encryption, NGFWs incorporate TLS termination capabilities, allowing the decryption, inspection, and re-encryption of HTTPS and other encrypted traffic flows [31].

Although NGFWs provide substantially richer inspection and policy capabilities, rule management and deployment may still require significant administrative coordination, and inspection-intensive functions such as DPI and TLS interception can introduce computational overhead. Appliance-centric deployment models can also be difficult to reconcile with highly distributed and rapidly changing cloud-native environments unless integrated with orchestration and policy-automation systems [13].

2.3.3. Limitations of Firewall Architectures in 5G Environments

Six limitations are particularly relevant when conventional rule-centric firewall deployments are applied to dynamic 5G/cloud-native environments:

Scalability constraints: Conventional rule-centric firewall deployments often assume a comparatively stable set of policy objects and endpoints. As the number of slices, services, and dynamically instantiated network functions increases, maintaining per-slice and per-service policy can cause substantial growth in ruleset size and state-management complexity [12].

Static policy management: In conventional deployments, policy management often relies on administratively maintained rules that may not automatically track rapidly changing CNF identities, addresses, or placements [12] [14].

Latency overhead: For stringent URLLC scenarios, additional inspection latency must be carefully controlled because the end-to-end/user-plane latency targets can be on the order of milliseconds. Security processing inserted into latency-sensitive paths must therefore consume only a controlled portion of the available latency budget [5] [31].

Centralised enforcement constraints: Traditional firewalls are deployed at fixed network boundaries, requiring all traffic to traverse a central inspection point. Routing all distributed edge traffic through a central inspection point can increase path length, create bottlenecks, and conflict with local-breakout objectives, motivating distributed or edge-local enforcement [28] [12].

Absence of context awareness: Conventional IP/port-based policies do not inherently include 5G-specific context such as S-NSSAI, SUPI, or session/service state; incorporating these attributes requires integration with 5G control and orchestration systems [15].

Manual incident response: Where policy updates depend on analyst-driven workflows, response time can be substantially longer than automated enforcement, motivating integration with automated detection and orchestration mechanisms, particularly given the speed at which volumetric attacks such as DDoS or GTP-U flooding can saturate network resources [16].

These limitations motivate investigation of firewall architectures that support dynamic configuration, richer contextual inputs, distributed enforcement, and automated security-response orchestration — the design space explored in this thesis.

2.4. Threat Landscape in 5G Networks

The architectural innovations introduced by 5G — service-based interfaces, network slicing, cloud-native deployment, and edge computing — collectively produce an attack surface that is qualitatively distinct from that of previous mobile network generations. This section categorises the principal threat classes relevant to 5G environments, organised according to the network layer and architectural component targeted.

Figure 2.2 summarises the four threat categories discussed in this section, mapping the architectural layer each targets (control-plane SBA, user-plane N3/N4, network slicing, and cloud/edge infrastructure) to the corresponding MITRE ATT&CK technique evaluated later in this thesis where such a mapping exists (Section 2.6.3).

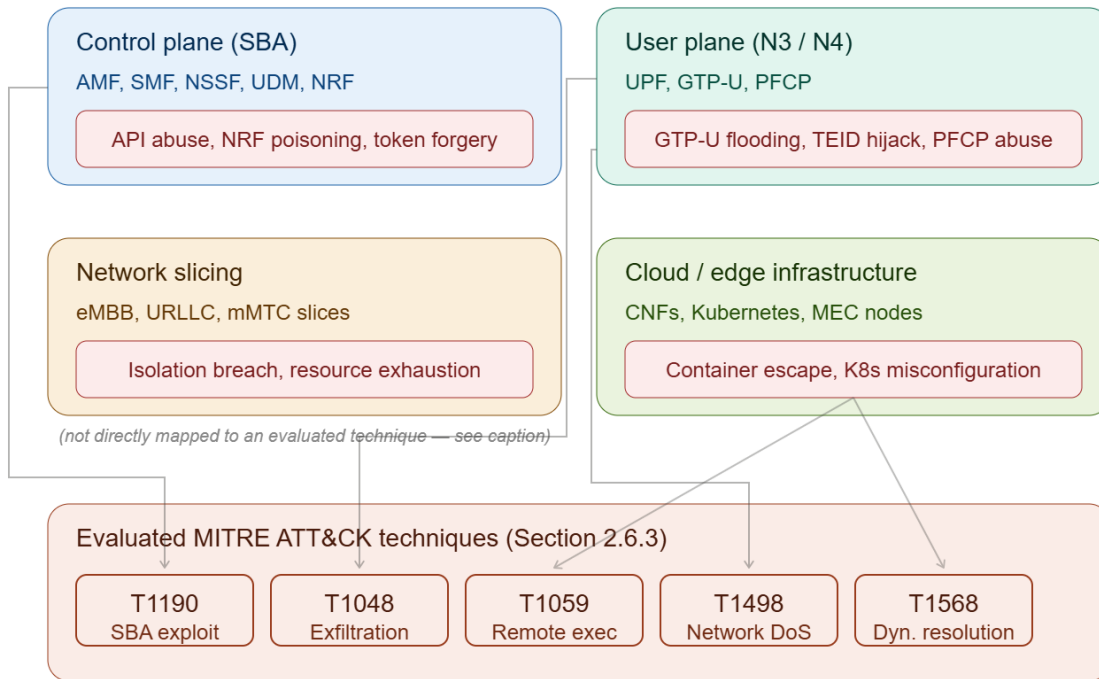


Figure 2.2: 5G threat landscape by architectural layer. Of the four threat categories discussed in Sections 2.4.1–2.4.4 — control-plane (SBA/API) attacks, user-plane (GTP-U/PFCP) attacks, network slice isolation threats, and cloud/edge infrastructure threats — three map directly to one or more of the five MITRE ATT&CK techniques evaluated in this thesis (Section 2.6.3): T1190 (SBA exploitation) to the control plane; T1048 (exfiltration, manifesting as bulk transfer over GTP-U tunnels) and T1498 (network denial of service) to the user plane; and T1059 (remote execution via SSH/Kubernetes) and T1568 (dynamic resolution for CNF command-and-control) to cloud/edge infrastructure. Network slice isolation threats (Section 2.4.3) are discussed in this chapter but do not correspond directly to any of the five techniques selected for empirical evaluation (Section 2.6.3); this is a scope limitation of the technique selection rather than an indication that slice-oriented threats are less significant. *Source: Author’s own diagram.*

2.4.1. Control Plane Attacks (SBA/API Abuse)

The use of HTTP/2 and web-style APIs exposes 5GC functions to classes of API-security threats that are already well established in web and cloud environments [10].

Unauthorised API Access: Network functions are required to present OAuth 2.0 access tokens when invoking services exposed by peer NFs. Weaknesses in token issuance, validation, or revocation mechanisms may permit an adversary to obtain or forge valid tokens, enabling unauthorised invocation of sensitive operations such as subscriber data retrieval from the UDM or session modification via the SMF [32].

Parameter Manipulation: API endpoints that accept subscriber or session identifiers as input parameters may be vulnerable to insecure direct object reference (IDOR) attacks, in which an adversary substitutes a legitimate identifier with that of another subscriber to access or modify unauthorised resources [10].

NRF Poisoning: The NRF serves as the service discovery registry for all 5GC network functions. An adversary capable of registering a rogue NF instance or modifying existing NF profiles within the NRF can redirect legitimate inter-NF communication to attacker-controlled endpoints, enabling man-in-the-middle interception or service disruption [33].

Injection Attacks: API endpoints that incorporate user-supplied input into backend queries or commands without adequate sanitisation are susceptible to injection attacks, including JSON injection and, where relational databases underlie subscriber management functions, SQL injection [10].

Denial of Service via API Flooding: The SBA's reliance on HTTP/2 connections between NFs introduces the possibility of request flooding attacks targeting specific NF instances. Exhausting the request handling capacity of a critical function such as the AMF or SMF can degrade network availability for all connected subscribers [16].

2.4.2. User Plane Attacks (DDoS and GTP-U Abuse)

The user plane, responsible for forwarding end-user data traffic, is exposed to a distinct set of threats primarily associated with the GTP-U tunnelling protocol employed on the N3 interface between the gNB and UPF [23].

- **GTP-U Flooding:** Volumetric denial-of-service attacks targeting the UPF via GTP-U packet flooding can saturate processing capacity, resulting in degraded throughput or complete unavailability of user plane services. The encapsulation overhead of GTP-U complicates the application of conventional rate-limiting mechanisms [23] [16].
- **GTP-U Spoofing and Injection:** GTP-U does not provide inherent source authentication in its base specification [23]. An adversary with access to the N3 network segment may inject forged GTP-U packets bearing arbitrary source addresses, potentially bypassing user plane security policies or injecting malicious content into user sessions.
- **Tunnel Endpoint Identifier (TEID) Manipulation:** Each GTP-U tunnel is identified by a TEID assigned during session establishment. An adversary who can enumerate or predict valid TEIDs may redirect user plane traffic belonging to legitimate sessions, enabling session hijacking or traffic interception [23].
- **Packet Fragmentation Attacks:** Deliberate fragmentation of GTP-U packets can impede stateful inspection mechanisms that require reassembly of complete packets prior to policy evaluation, potentially enabling the evasion of detection systems [23].
- **PFCP Protocol Abuse:** The N4 interface between the SMF and UPF employs the Packet Forwarding Control Protocol (PFCP) to install and modify packet detection and forwarding rules. Unauthorised or malformed PFCP messages targeting the UPF

can alter its forwarding behaviour, redirect traffic, or disable enforcement of QoS and security policies [35].

A note on empirical scope. The proof-of-concept dataset used later in this thesis (Chapter 6) does not reproduce the GTP-U, PFCP, or SBA protocols described above directly; it models selected behavioural characteristics — such as volumetric flooding and bulk data transfer — at the aggregate flow level, using synthetic traffic that is protocol-agnostic. This distinction is restated in Section 6.1 and should be kept in mind when interpreting the technique-to-5G mappings introduced in Section 2.6.3 below.

2.4.3. Network Slice-Oriented Threats

Network slicing enables the simultaneous operation of multiple logically isolated virtual networks on shared physical infrastructure. While slicing provides significant operational flexibility, the isolation boundaries between slices represent a critical security requirement whose violation can have severe consequences [34].

- **Inter-Slice Isolation Breach:** The most severe slice-related threat involves an adversary escaping the boundaries of one slice and gaining unauthorised access to resources or traffic belonging to another. Depending on the slices involved, such a breach could expose sensitive subscriber data, compromise critical infrastructure services, or enable lateral movement within the operator’s infrastructure [34].
- **Resource Exhaustion Attacks:** Shared physical resources — including CPU, memory, network bandwidth, and storage — underlie all network slices. An adversary operating within one slice may deliberately generate workloads that consume disproportionate shares of these resources, degrading the performance or availability of co-resident slices, a threat analogous to the noisy neighbour problem in cloud computing [34].
- **NSSF Manipulation:** The Network Slice Selection Function determines slice assignment for each UE based on subscription data and operator policy. Attacks targeting the NSSF or the interfaces through which slice selection information is communicated may result in UEs being assigned to incorrect slices, potentially exposing them to unauthorised network environments or denying them access to services for which they are entitled [34].
- **Slice-Specific Configuration Exploitation:** Individual slices may be configured with distinct security policies reflecting their service requirements. A slice optimised for low-latency IoT connectivity, for example, may apply less stringent security controls than one serving enterprise users. Adversaries may selectively target slices with weaker security postures as entry points for broader attacks [34].

2.4.4. Cloud and Edge Infrastructure Threats

The deployment of 5G network functions on cloud-native infrastructure and at the network edge introduces threats originating from the underlying compute and orchestration layers [11].

Container Escape: A vulnerability in the container runtime or Linux kernel may permit a process executing within a container to break out of its isolation boundary and gain access to the host operating system. From this privileged position, an adversary can access the file systems, network interfaces, and process spaces of all co-resident containers, effectively compromising all CNFs deployed on the affected host [11].

Malicious Container Images: CNFs are deployed from container images that may be sourced from public or private registries. An adversary who compromises an image repository or introduces a backdoored image into the supply chain can achieve persistent access to any environment in which the image is subsequently deployed [11].

Supply Chain Attacks: 5G CNFs depend on software libraries and dependencies that may themselves be compromised. A supply chain attack targeting a widely used cryptographic library or network protocol implementation can affect a large number of CNF deployments simultaneously [11].

Kubernetes Control Plane Exploitation: The Kubernetes API server, if exposed without adequate authentication and authorisation controls, provides an adversary with the capability to deploy arbitrary workloads, exfiltrate secrets, modify network policies, and disrupt the operation of all managed CNFs. Misconfiguration of RBAC policies is a particularly prevalent source of unintended privilege escalation [26].

Physical Security of Edge Nodes: Edge computing nodes are frequently deployed in locations with limited physical security controls, such as street cabinets, rooftop installations, or shared colocation facilities. Physical access to an edge node may enable an adversary to extract cryptographic material, install persistent implants, or disrupt local services [28].

Workload Migration Vulnerabilities: The dynamic migration of CNF instances between edge nodes, performed in response to user mobility or resource availability, requires the secure transfer of application state and the consistent application of security policies at the destination node. Failures in this process may create transient windows during which security controls are not enforced [28].

2.5. Model Context Protocol (MCP)

The Model Context Protocol (MCP) is an open standard developed by Anthropic that defines a structured communication interface between LLM-based agents and external tools, data sources, and services [8]. Prior to the introduction of MCP, the integration of LLMs with external systems required bespoke, application-specific implementations that were difficult to reuse, audit, or extend. MCP addresses this limitation by specifying a uniform protocol through which an LLM agent can discover available tools, invoke them with structured arguments, and receive structured responses, independent of the underlying implementation of either the agent or the tool provider.

2.5.1. Architecture and Components

Figure 2.3 illustrates the flow of a user interaction through the MCP host, client, and server, to the underlying data sources exposed as tools.

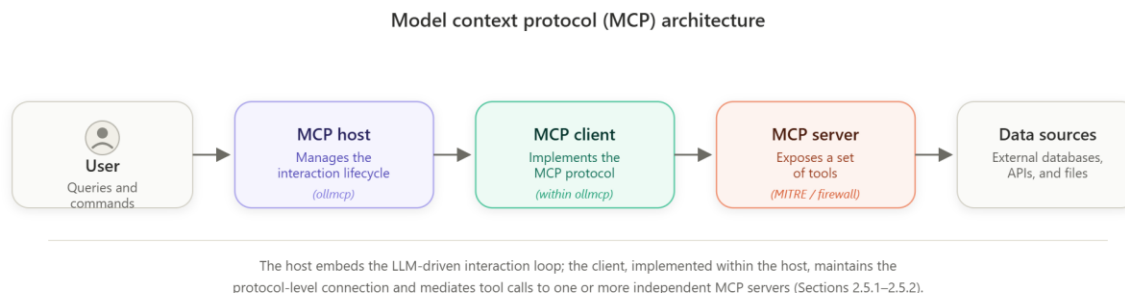


Figure 2.3: MCP architecture and interaction flow. A user’s queries and commands are received by the MCP host, which embeds the LLM-driven interaction loop; the MCP client, implemented within the host, maintains the protocol-level connection to one or more independent MCP servers, each of which exposes a set of tools backed by underlying data sources (e.g., APIs, files, or — in this thesis — the MITRE ATT&CK knowledge base and the firewall enforcement interface). *Source: Author’s own diagram.*

The MCP architecture comprises three principal components: the MCP host, the MCP client, and the MCP server [8]. The MCP host is the application that embeds the LLM-driven interaction loop and manages the overall interaction lifecycle; the MCP client is the component, typically embedded within the host, that implements the MCP protocol itself — maintaining connections to MCP servers and mediating tool invocation requests.

In the implementation used in this thesis (Section 5.4), the host and client roles are both realised by `ollmcp`. The evaluation pipeline invokes `ollmcp`, which manages the LLM-driven interaction loop and maintains MCP connections to the two servers described below. Ollama acts only as the remote model-inference backend that `ollmcp` connects to (via the `--host` and `--model` arguments, Appendix A.1) in order to obtain the LLM’s reasoning and tool-call decisions; Ollama is not itself the MCP host or client. This distinction correctly locates the model-agnostic property discussed in Section 4.5 at the `ollmcp`/MCP-client boundary, rather than treating the specific inference backend as part of the MCP architecture itself.

The MCP server is an independent process that exposes a set of capabilities — tools, resources, and prompts — through the MCP interface. Each server is responsible for a specific domain of functionality. In this thesis, two MCP servers are implemented: one exposing MITRE ATT&CK threat intelligence functions, and one exposing firewall enforcement functions via `iptables`.

Communication between the MCP client and MCP servers follows a request-response model over a transport layer. The `stdio` transport, used in this thesis, communicates via

standard input and output streams, making MCP servers deployable as local subprocess invocations without requiring network configuration.

2.5.2. Tool Invocation Model

The primary interaction primitive in MCP is the tool. Each MCP server declares a set of named tools, each described by a name, a natural language description, and a JSON Schema specification of its input parameters [36]. When an LLM agent determines that a tool invocation is appropriate — based on its reasoning about the current task and the available tool descriptions — it generates a structured tool call specifying the tool name and argument values. The MCP client forwards this call to the appropriate server, which executes the corresponding function and returns a structured result.

This model provides several properties relevant to security automation. Tool descriptions in natural language allow the LLM to reason about which tool is appropriate for a given situation without requiring explicit programming of tool selection logic. The JSON Schema parameter specification enables validation of the structure of LLM-generated arguments before execution, though schema validation alone does not address semantic constraints, authorisation, or command-injection risks at the enforcement boundary — these are addressed separately in Chapter 8. The separation between the LLM (which reasons about what to do) and the MCP server (which executes actions) enables the interposition of authorisation checks and audit logging at the server boundary.

In this thesis, the following tools are exposed through MCP:

Through the MITRE MCP server: `get_technique_info` (retrieve technique description by ID), `get_balanced_mitigations` (retrieve network-relevant mitigations for a technique), and `full_security_analysis` (combined technique and mitigation retrieval).

Through the firewall MCP server: `block_ip` (apply an iptables DROP rule for a source address), `unblock_ip` (remove an existing block rule), `apply_mitre_mitigation` (block an attacker IP in the context of a specific MITRE technique), `list_blocked` (enumerate currently active block rules), and `get_firewall_stats` (retrieve enforcement statistics).

2.5.3. Security Considerations for MCP

The delegation of executable actions to an LLM agent via MCP introduces or amplifies security considerations that are less prominent in deterministic automation pipelines [4]. Four principal risks are relevant to the system described in this thesis.

Prompt injection refers to the possibility that attacker-controlled content present in data retrieved by the LLM — for example, within MITRE technique descriptions — could influence the LLM's subsequent reasoning and cause it to generate unintended tool calls. An adversary who can modify the content of a MITRE technique description, or who can influence the data pipeline feeding detection events to the LLM, may be able to manipulate the agent's behaviour.

LLM hallucination refers to the generation of plausible but incorrect outputs by the LLM. In the present architecture, hallucination could cause incorrect technique interpretation,

selection of an inappropriate mitigation action, or invocation of an enforcement tool against an incorrect target. Because actual iptables syntax is generated by predefined server-side templates rather than free-form LLM output (Section 5.3), syntactically invented firewall commands are constrained by the enforcement interface, though incorrect *target* selection remains possible.

Unauthorised tool invocation refers to the possibility that the LLM agent generates tool calls that exceed its intended authorisation scope — for example, invoking `unblock_ip` to remove existing protective rules, or invoking enforcement actions against addresses outside the intended target set.

Tool or server compromise (tool poisoning) refers to the risk that a malicious or compromised MCP server returns tool descriptions or results crafted to manipulate the orchestrating LLM’s behaviour, since the LLM typically has no independent means of verifying that a connected MCP server is legitimate or unmodified. This risk is distinct from prompt injection via retrieved *content* (above), as it concerns the trustworthiness of the tool-providing server itself.

These risks are addressed through the human-in-the-loop mechanisms and additional mitigations discussed in Chapter 8.

2.6. MITRE ATT&CK Framework

The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework is a continuously maintained knowledge base of adversary behaviours observed in real-world attacks against enterprise, mobile, and industrial control system environments [7]. Developed and maintained by the MITRE Corporation, the framework has become a de facto standard for threat intelligence, security assessment, and defensive gap analysis in both industry and academic research [7].

2.6.1. Structure and Organisation

The ATT&CK framework organises adversary behaviour into a three-level hierarchy: tactics, techniques, and sub-techniques [7].

Tactics represent the adversary’s high-level objectives at a given stage of an intrusion, such as Initial Access, Execution, Persistence, Privilege Escalation, Defence Evasion, Credential Access, Discovery, Lateral Movement, Collection, Command and Control, Exfiltration, and Impact. Each tactic answers the question of why an adversary performs an action.

Techniques describe the specific methods by which an adversary achieves a tactical objective. Each technique is assigned a unique identifier (e.g., T1190 for Exploit Public-Facing Application) and is documented with a description of adversary behaviour, detection guidance, and recommended mitigations. As of the Enterprise ATT&CK release used in this thesis (MITRE ATT&CK Enterprise, `enterprise-attack.json`, retrieved 30 March 2026), the framework documents over 200 techniques across 14 tactics.

Sub-techniques provide additional specificity below the technique level, describing distinct variants of a parent technique. Sub-technique identifiers extend the parent identifier with a numeric suffix (e.g., T1190.001).

2.6.2. STIX 2.0 Representation

The ATT&CK knowledge base is distributed in Structured Threat Information eXpression (STIX) 2.0 format, a JSON-based standard for representing cyber threat intelligence [37]. This is confirmed directly from the downloaded `enterprise-attack.json` bundle used in this thesis (Section 5.2, Appendix A.2), whose root-level `spec_version` field explicitly states "2.0" — consistent with the `mitreattack-python` library (version 6.1.0) module used to parse it, `mitreattack.stix20` (Appendix C.1). In the STIX representation, ATT&CK techniques are encoded as `attack-pattern` objects, mitigations as `course-of-action` objects, and the relationships between them as `relationship` objects with a `mitigates` relationship type.

The `mitreattack-python` library, used in the MITRE MCP server implementation described in Chapter 5, provides a high-level Python API for querying the ATT&CK STIX bundle. It supports retrieval of technique objects by ATT&CK identifier, enumeration of mitigations associated with a technique, and traversal of technique-to-software and technique-to-group relationships.

2.6.3. Techniques Relevant to 5G Environments

Five ATT&CK Enterprise techniques are selected for evaluation in this thesis, chosen on the basis of their relevance to the 5G threat landscape described in Section 2.4. Table 2.1 lists each identifier alongside its official MITRE ATT&CK name and the traffic interpretation used in this thesis. The technique-to-5G mappings below describe plausible 5G manifestations of each technique for the purposes of this thesis's evaluation; they are not official ATT&CK-to-5G mappings published by MITRE, and readers should interpret them as illustrative applications of generic ATT&CK techniques to the 5G context considered here, consistent with the empirical-scope note in Section 2.4.2.

Table 2.1: MITRE ATT&CK identifiers used in this thesis

ID	Official ATT&CK Name	Thesis Traffic Interpretation
T1190	Exploit Public-Facing Application	Synthetic SBA/API exploitation
T1048	Exfiltration Over Alternative Protocol	Synthetic exfiltration
T1059	Command and Scripting Interpreter	Synthetic remote execution
T1498	Network Denial of Service	Synthetic distributed DoS
T1568	Dynamic Resolution	Synthetic dynamic-resolution behaviour

T1190 — Exploit Public-Facing Application: Adversaries exploit vulnerabilities in internet-facing applications to gain initial access. In the context considered in this thesis, a plausible 5G manifestation is the exploitation of SBA API endpoints exposed by network functions such as the AMF, SMF, and NRF [7].

T1048 — Exfiltration Over Alternative Protocol: Adversaries exfiltrate data using protocols other than the primary command-and-control channel, often to evade detection.

A plausible 5G manifestation is bulk data transfer over non-standard ports or encapsulated within GTP-U tunnels [7]; the synthetic dataset used in this thesis does not reproduce GTP-U encapsulation directly (Section 2.4.2) but represents the volumetric/behavioural signature of such transfer at the flow level.

T1059 — Command and Scripting Interpreter: Adversaries use command-line interfaces or scripting environments to execute malicious commands on compromised systems. A plausible 5G manifestation, in cloud-native deployments, is post-compromise execution within containerised network functions via SSH or the Kubernetes API [7].

T1498 — Network Denial of Service: Adversaries perform network DoS attacks to degrade or block the availability of targeted resources by exhausting network bandwidth or connection-handling capacity. In the context of this thesis, T1498 is used as a generic representation of distributed network DoS targeting 5G network functions, consistent with the user plane flooding threats described in Section 2.4.2; the synthetic scenario does not reproduce GTP-U flooding directly [7].

T1568 — Dynamic Resolution: Adversaries use dynamic DNS or domain generation algorithms to evade static blocking of command-and-control infrastructure. A plausible 5G manifestation is CNFs that establish outbound connections to attacker-controlled infrastructure via dynamically resolved domains [7].

2.7. Chapter Summary

This chapter established the technical foundations for the proposed architecture. The 5G Core, cloud-native deployment model, and MEC introduce dynamic and distributed security requirements that challenge conventional rule-centric firewall management (Sections 2.1–2.3). Section 2.4 categorised the resulting 5G threat landscape across control-plane, user-plane, network-slicing, and cloud/edge domains. The chapter also introduced MCP as the tool-coordination mechanism used by the LLM orchestrator (Section 2.5) and MITRE ATT&CK as the structured threat-intelligence source used by the proposed pipeline (Section 2.6). Five ATT&CK techniques were selected as representative scenarios for the proof-of-concept evaluation. Chapter 3 next positions this architecture relative to existing intrusion-detection, programmable-firewall, LLM-security, and MCP research.

3. Related Work

This chapter situates the present work within four principal bodies of prior research: machine-learning-based intrusion detection (Section 3.1), adaptive and programmable firewall architectures (Section 3.2), LLM-based security automation (Section 3.3), and MCP-based systems and security (Section 3.4). Two additional areas relevant to the 5G applicability of the proposed architecture are reviewed: machine learning for 5G and network-slice security (Section 3.5) and federated learning for distributed intrusion detection (Section 3.6). Section 3.7 synthesises these findings into a gap analysis that positions the contributions of this thesis (Section 1.6) relative to this prior work.

3.1. Machine Learning and Deep Learning-Based Intrusion Detection

A substantial body of work applies machine learning and deep learning to network intrusion detection, motivated in part by the limitations of signature-based systems when confronted with previously unseen or previously unmodelled attack behaviours.

Mirsky et al. proposed Kitsune, an unsupervised, online network intrusion detection system built around KitNET, an ensemble of autoencoders that incrementally learns normal traffic patterns from packet-level statistics and flags deviations as anomalies [2]. Kitsune is notable for operating without labelled training data and for its lightweight design, which the authors demonstrate can run on resource-constrained hardware such as a Raspberry Pi [2]. This anomaly-based, unsupervised paradigm contrasts with the rule-based detection approach adopted in this thesis (Section 4.2): whereas Kitsune learns a model of normal behaviour and flags statistical deviation, the detection engine in this thesis applies fixed, dataset-calibrated quantitative thresholds that are mapped to specific MITRE ATT&CK technique labels. As an anomaly-based approach, Kitsune is designed to detect deviations that need not correspond to previously labelled attack classes, potentially providing broader coverage of previously unseen behaviours than fixed signature-derived rules. However, it does not directly produce an ATT&CK technique identifier suitable for the downstream retrieval process used in this thesis.

Du et al. proposed DeepLog, which applies a Long Short-Term Memory (LSTM) network to model system log sequences as a natural-language-like stream, learning the normal execution order of log keys and flagging deviations from the learned sequence model as anomalies [3]. DeepLog further supports incremental online updating and constructs workflow representations that assist root-cause diagnosis once an anomaly has been detected [3]. While DeepLog operates on host-level system logs rather than network traffic, it shares with the present work a concern for translating a detection event into actionable diagnostic information; however, DeepLog does not extend this translation into automated enforcement, nor does it draw on an external structured threat-intelligence framework such as MITRE ATT&CK.

Closer to the per-source aggregate traffic-feature classification setting evaluated in this thesis, a substantial body of work has benchmarked supervised and ensemble classifiers — including random forests, decision-tree variants, support vector machines, and deep architectures — on datasets such as CICIDS2017 and CSE-CIC-IDS2018. Rodríguez et

al. systematically compare ten classification techniques on CICIDS2017 flow-level features, finding that decision-tree-based methods (random forest, PART, J48) consistently achieve the highest F1 scores (above 0.999 on the complete dataset) for both binary and multi-class attack classification, and that correlation-based feature selection can substantially reduce inference time with only a small performance cost [52]. Other studies report comparable or higher accuracy using deep architectures (e.g., LSTM- or CNN-based classifiers) on the same or related benchmark datasets, though typically at greater computational cost than tree-based methods [38]. These approaches report high accuracy on benchmark datasets but, as with the evaluation presented in Chapter 7 of this thesis, are subject to concerns regarding the representativeness of benchmark traffic relative to operational networks and the risk of dataset-specific overfitting — concerns that are discussed in detail for the present system in Section 7.2.

Across this body of work, a common limitation relevant to the present thesis is that a substantial portion of ML/DL-based IDS research evaluates detection or classification performance as its primary endpoint, while automated translation of the resulting classification into threat-intelligence-informed enforcement is less commonly evaluated as part of the same end-to-end system. The present thesis investigates this gap through a proof-of-concept pipeline in which technique-specific detection outputs are connected to ATT&CK-informed mitigation reasoning and live network enforcement, for the subset of attack categories aligned with MITRE ATT&CK techniques.

3.2. Adaptive, Programmable, and Automated Security Enforcement

A second relevant body of work concerns the evolution of firewall and security enforcement mechanisms away from the static, appliance-centric model described in Section 2.3, toward programmable, software-defined, and increasingly automated enforcement.

3.2.1. SDN and eBPF-Based Enforcement

Software-Defined Networking (SDN) enables centralised control of network forwarding behaviour through a programmable control plane, and a substantial literature has explored SDN-based firewalls in which a central controller installs and updates flow rules across distributed switches in response to detected conditions [39]. SDN firewalls address the static-policy limitation identified in Section 2.3.3 by enabling rule updates to be pushed network-wide from a single point of control; however, the translation from a detected threat to a specific flow rule in SDN firewall literature is typically performed by application-specific controller logic, rather than by a general-purpose reasoning component capable of consulting an external threat-intelligence knowledge base. This closely related body of work demonstrates that automated, programmatic rule deployment following detection is well established in the literature and is therefore not itself a contribution of this thesis; the contribution of the present thesis is instead the MITRE ATT&CK-driven interpretation and MCP-mediated reasoning layer interposed between detection and enforcement.

More recently, eBPF-based dataplanes — most prominently Cilium — have emerged as an alternative to iptables-based enforcement in cloud-native environments [40]. Cilium implements Kubernetes network policy directly in the Linux kernel via eBPF, associates policy with workload identity rather than IP address, and supports identity-aware and Layer 7-aware policy enforcement for protocols such as HTTP and gRPC [40]. This identity-based, kernel-level enforcement model addresses several of the scalability and context-awareness limitations of iptables-based firewalls discussed in Section 2.3.3, and is directly relevant to the 5G applicability discussion in Chapter 9: a production deployment of the architecture proposed in this thesis would plausibly target a Cilium-based enforcement layer rather than the single-node iptables enforcement used in the present proof-of-concept (Section 4.4). However, the Cilium and eBPF literature surveyed here addresses the enforcement substrate itself and does not incorporate LLM-based reasoning or external threat-intelligence integration; it is therefore complementary to, rather than overlapping with, the contribution of this thesis.

3.2.2. Intent-Based and Automatic Policy Refinement

A further related body of work is intent-based networking (IBN) and automatic security policy refinement, in which operators or analysts specify desired outcomes at a high level and an automated system translates these intents into low-level configuration [41]. Colaiacomo et al.’s RefinementEngine exemplifies this direction for security specifically: it translates high-level intent into device-level filtering policy, producing deployable firewall rules under network constraints via constraint solving, rather than LLM-based reasoning [14]. The proposed pipeline in this thesis (Section 4.6) shares a conceptual similarity with this line of work: higher-level defensive guidance is interpreted and translated into a lower-level enforcement action. However, MITRE ATT&CK mitigation guidance is not itself a formal network intent in the IBN sense — it is defensive knowledge rather than a specification of desired network state — and the present architecture does not implement a complete intent-validation or assurance loop of the kind found in mature IBN systems. The distinction is therefore one of both mechanism (constraint-solving or declarative compilation versus LLM-based reasoning) and input type (a formal intent specification versus natural-language threat-intelligence guidance).

3.2.3. Security Orchestration, Automation, and Response (SOAR)

Security Orchestration, Automation, and Response (SOAR) platforms already implement an end-to-end pipeline conceptually close to the one proposed in this thesis: an alert is enriched with contextual and threat-intelligence data, matched against a playbook, and translated into an enforcement or remediation action across one or more heterogeneous security tools [17]. Islam et al.’s multi-vocal review of security orchestration documents this alert-enrichment-playbook-enforcement pattern across both commercial and open-source SOAR platforms, and identifies the lack of interoperability among the underlying security tools as a persistent obstacle to fully automating this pipeline [17]. The present thesis’s architecture can be read as addressing a specific instance of this interoperability problem — using MCP as the tool-coordination layer — while additionally introducing LLM-mediated reasoning into the enrichment-to-action translation process rather than relying exclusively on fixed, pre-authored playbook logic, and using MITRE ATT&CK

specifically (rather than generic threat-intelligence feeds) as the structured knowledge source consulted during that translation. Traditional SOAR architectures primarily rely on pre-authored playbooks for enrichment-to-action translation, although contemporary platforms increasingly incorporate LLM-assisted steps. The architecture investigated in this thesis instead explicitly places MCP-mediated LLM reasoning over ATT&CK-derived context within the enforcement workflow.

It is also worth briefly situating this comparison relative to existing cybersecurity interoperability standards, since MCP is presented in this thesis (Section 2.5) as addressing heterogeneous tool coordination. STIX [37] defines a structured representation for cyber threat intelligence, TAXII [55] provides mechanisms for exchanging that intelligence between systems, and OpenC2 [56] defines a command language for directing cyber-defence actuators. These standards solve data-exchange and actuator-command interoperability, which is a different layer of the problem than the agent-tool invocation interface that MCP provides for LLM-mediated reasoning; the two are complementary rather than competing.

3.3. LLM-Based Security Automation

The application of large language models to cybersecurity tasks has expanded rapidly, with particular attention to penetration testing and tool-augmented agents.

Deng et al. introduced PentestGPT, an LLM-empowered automated penetration testing tool structured around three cooperating modules — reasoning, generation, and parsing — designed to mitigate the context-loss problems that arise when a single LLM session is used to drive a multi-step penetration test [22]. The authors report substantial improvements in task completion relative to using an LLM directly, and the tool has subsequently been extended with typed tool interfaces and difficulty-aware planning in later work [42]. PentestGPT and its successors demonstrate the general viability of decomposing security workflows into LLM-driven sub-tasks coordinated through structured tool interfaces — an architectural pattern that the MCP-based pipeline in this thesis (Section 4.5–4.6) also follows, albeit applied to defensive rather than offensive operations, and with a narrower, fixed tool surface (MITRE lookup and firewall enforcement) rather than a broad penetration-testing toolchain.

A related line of work, exemplified by PentestAgent, employs multi-agent architectures combined with retrieval-augmented generation (RAG) to integrate external knowledge sources into LLM-driven security workflows [43]. This retrieval-augmentation pattern is conceptually similar in intent to the role played by the MITRE MCP server in this thesis (Section 4.3): both approaches aim to ground LLM reasoning in an external, structured knowledge source rather than relying solely on the model’s parametric knowledge. The two mechanisms are not universally equivalent, however: in a conventional non-agentic RAG pipeline, retrieved content is typically supplied to the model by an external retrieval component operating independently of the model’s own decision-making. MCP, by contrast, exposes capabilities through an explicit agent-tool interface; depending on the orchestration pattern, these tools may be invoked deterministically by the host or selected

dynamically by the model (Section 2.5.2) — a distinction that motivates the agentic MCP evaluation presented in Section 7.7.

A prominent strand of LLM cybersecurity research has focused on offensive automation, particularly penetration testing, while defensive applications increasingly address security-operations-centre (SOC) workflows: Saju and Azim present an end-to-end LLM framework for SOC threat detection, query generation, and incident resolution, reporting reductions in incident-triage time in their evaluated setting [53], and Habibzadeh et al. survey the broader emerging literature on LLM integration into SOC workflows, covering log analysis, alert triage, and threat-intelligence-informed explanation [54]. Fewer studies, however, evaluate LLM-mediated modification of live network-enforcement state as part of an end-to-end defensive pipeline — the capability that distinguishes the firewall enforcement MCP server described in Section 4.4 and that motivates the security analysis in Chapter 8.

3.4. MCP-Based Systems and MCP Security

The Model Context Protocol, introduced by Anthropic in late 2024, defines an open protocol and interface through which LLM agents discover and invoke external tools (Section 2.5) [8]. Given its recency, the academic literature specifically addressing MCP-based systems is limited but growing rapidly, and is concentrated heavily on security analysis rather than on application-domain case studies such as the one presented in this thesis. Several of the works cited in this section are recent preprints rather than peer-reviewed publications, reflecting the pace at which this literature is emerging; their publication status is noted where relevant.

Several works have systematically catalogued MCP-specific security risks. Hou et al. provide an early landscape analysis of MCP security threats and identify research directions for securing agent-tool interactions [4]. Guo et al. present a systematic analysis of MCP security covering the protocol’s trust model and the threats introduced when tool outputs are treated as trusted context [44]. Subsequent work has proposed defence frameworks such as MCP-Guard, a multi-stage defence-in-depth framework for agentic MCP deployments [45], and MCPShield, which focuses on adaptive trust calibration for MCP agents [46]. A further strand of this literature addresses “tool poisoning” attacks, in which a malicious or compromised MCP server returns tool descriptions or outputs crafted to manipulate the orchestrating LLM’s behaviour [47] — a threat directly analogous to the prompt-injection-via-MITRE-description scenario analysed in Section 8.2 of this thesis, and to the compromised-MCP-server scenario analysed in Section 8.6.

The MCP security literature reviewed here focuses primarily on general-purpose agent/tool security and abstract or illustrative threat scenarios. Within the reviewed literature, no worked case study was identified in which an MCP-mediated LLM agent connects structured cybersecurity threat intelligence to a live network-enforcement mechanism. The architecture proposed in this thesis (Chapter 4) and its accompanying security analysis (Chapter 8) can therefore be read as a concrete, domain-specific instantiation of the general MCP security concerns raised in this literature. The literature reviewed for this thesis did not identify prior work combining MITRE ATT&CK-driven mitigation retrieval with MCP-mediated live network enforcement; iptables is used in this

thesis as the specific enforcement backend for this proof-of-concept (Section 4.4), not as the locus of the identified gap.

3.5. Machine Learning for 5G and Network Slice Security

This literature is reviewed primarily to contextualise the architectural applicability and future extensions of the proposed system discussed in Section 9.4; the proof-of-concept evaluated in this thesis does not implement slice-aware detection or evaluate traffic against a real 5G Core (Section 6.1). A separate strand of research applies machine learning specifically to the security challenges of 5G architectures introduced in Section 2.4, particularly network slicing.

Dangi et al. provide a comprehensive survey of machine learning-based approaches to 5G network slicing security, covering the application of ML, SDN, and NFV across the slice lifecycle and cataloguing the attack types and corresponding ML-based defences proposed in the literature [51]. More recent work has proposed slice-specific intrusion detection models, evaluating different ML algorithms for the eMBB, mMTC, and URLLC slice types and reporting that algorithm choice should be tailored to the latency and accuracy requirements of each slice — for example, favouring computationally lighter models such as Random Forest for the latency-sensitive URLLC slice [48]. Other work has focused specifically on ML-based detection of IoT-driven DDoS attacks within the 5G Core, emphasising feature-selection techniques that reduce computational overhead to meet stringent latency targets for selected URLLC scenarios [49].

This body of work establishes two points of direct relevance to the present thesis. First, it confirms that latency constraints — particularly for the URLLC slice class — are treated in the literature as a primary design constraint for any ML-based security mechanism, reinforcing the importance of the latency evaluation performed in Section 7.4 for latency-sensitive ML-based security mechanisms specifically. Second, it indicates that slice-aware detection — in which the detection model or its thresholds are conditioned on the network slice to which traffic belongs — is an established direction that the present thesis’s proof-of-concept does not address, since the synthetic dataset (Chapter 6) contains no slice-identifying information. This is noted as a direction for future work in Section 9.6.

3.6. Federated Learning in Network Security

Federated learning (FL) has been widely proposed as a means of training intrusion detection models across distributed network domains — such as separate operator deployments, edge sites, or IoT gateways — without centralising raw traffic data, thereby addressing both data-privacy and bandwidth constraints [50]. Surveys of FL-based intrusion detection systems describe a taxonomy spanning learning and aggregation strategies, client topologies, and detection paradigms (signature-based, anomaly-based, and hybrid), and identify communication overhead, data heterogeneity across clients, and robustness to poisoning attacks on the aggregation process as ongoing challenges [50]. Several studies surveyed in [50] report performance comparable to centrally trained models under their respective evaluation settings, while avoiding the transmission of raw traffic, and FL-based architectures have been proposed specifically for industrial and IoT

intrusion detection contexts structurally similar to the multi-slice, multi-tenant character of 5G Core deployments [50].

The relevance of this literature to the present thesis is primarily architectural rather than direct: the rule-based detection engine evaluated in Chapter 6 involves no model training and therefore no federated learning component. However, the FL-IDS literature is directly relevant to the 5G deployment model discussed in Section 9.4, since a production deployment of the proposed architecture across multiple network slices or edge sites could benefit from federated learning where raw traffic cannot or should not be centralised across slices, sites, or administrative domains — one of several plausible approaches (alongside centrally trained, transfer-learned, or independently trained local models) to preserving the isolation properties required between slices (Section 2.4.3), should the fixed-threshold rules used in this proof-of-concept be replaced with a learned detection component in future work. This thesis does not evaluate a federated learning component, and its absence is noted explicitly as a scope limitation in Section 9.3.

3.7. Gap Analysis

Table 3.1 summarises the positioning of the present thesis relative to the bodies of related work reviewed above, along four dimensions corresponding to the core components of the proposed architecture (Chapter 4). For this comparison: **structured ATT&CK integration** denotes runtime retrieval or use of MITRE ATT&CK knowledge within the decision pipeline itself (rather than mere topical relevance to security); **LLM-based reasoning** denotes use of an LLM to interpret context, classify, or select actions, as opposed to purely rule-based or statistical decision logic; **MCP-based coordination** denotes use of MCP specifically as the agent-tool interface, rather than generic or ad hoc function-calling; and **automated live network enforcement** denotes automated modification of an active network-policy, firewall, or security-group state, rather than detection, alerting, or human-operated response alone.

Table 3.1: Gap analysis — positioning of this thesis relative to related work

Table entries characterise the representative works cited for each category rather than asserting that every publication within that research area shares the listed property.

Research category	Representative refs	Structured ATT&CK Integration	LLM-Based Reasoning	MCP-Based Coordination	Automated Live Network Enforcement
ML/DL-based IDS	[2], [3], [38], [52]	No	No	No	No (detection/diagnosis only)
SDN / eBPF-based enforcement	[39], [40]	No	No	No	Yes (policy push, not threat-intel-driven)
Intent-based / automatic policy refinement	[14], [41]	No	No (constraint-solving or declarative compilers)	No	Yes (intent-driven, not LLM-mediated)
SOAR / security orchestration	[17]	Platform-dependent (typically enrichment/mapping rather than runtime LLM reasoning)	Partial (some platforms embed LLM steps)	No	Yes (orchestrated across heterogeneous tools)
LLM-based security agents	[22], [42], [43], [53], [54]	No (general security knowledge)	Yes	No (tool-augmented via	Limited (mostly offensive tooling or

Research category	Representative refs	Structured ATT&CK Integration	LLM-Based Reasoning	MCP-Based Coordination	Automated Live Network Enforcement
(offensive and defensive)				non-MCP or RAG-based interfaces)	advisory)
MCP security research	[4], [44], [45], [46], [47]	No	Varies (LLM agency is typically the subject of security analysis rather than an implemented reasoning workflow)	Yes	No (illustrative/abstract scenarios)
5G slice-aware ML/IDS	[48], [49], [51]	No	No (traditional ML)	No	No (detection only)
Federated learning for IDS	[50]	No	No	No	No (detection only)
This thesis	—	Yes	Yes	Yes	Yes (iptables, MCP-mediated)

As Table 3.1 illustrates, no single research category combines all four properties. ML/DL-based IDS research (Section 3.1) and 5G slice-aware ML approaches (Section 3.5) provide strong detection methodologies but stop short of automated enforcement and do not incorporate structured threat-intelligence frameworks.

The categories closest to the present thesis’s enforcement goal — SDN/eBPF enforcement, intent-based policy refinement, and SOAR (Section 3.2) — demonstrate that automated, programmatic translation from a detected condition to a live enforcement action is already a well-established capability and is not itself the contribution of this thesis. The representative works reviewed in these categories do not demonstrate the same combination of runtime LLM-mediated reasoning over structured ATT&CK context and MCP-based live network enforcement; their translation logic is instead primarily pre-authored, declarative, or application-specific (SOAR playbooks, IBN compilers, or SDN controller code).

LLM-based security automation (Section 3.3) and the MCP security literature (Section 3.4) establish, respectively, the viability of LLM-driven reasoning in security workflows and the risks of MCP-mediated tool orchestration, but the literature reviewed for this thesis did not identify prior work combining MITRE ATT&CK-driven mitigation retrieval with MCP-mediated live network enforcement within a single, experimentally evaluated pipeline. The contribution of this thesis (Section 1.6) is therefore best understood not as an improvement to any single one of these bodies of work in isolation, but as an integration across all four core properties — together with an empirically grounded account (Chapter 7) of the latency overhead and validity limitations observed in the evaluated implementation.

The thesis additionally investigates a question not directly evaluated by the works reviewed above (Sections 7.6–7.7): whether an LLM supplied with per-source aggregate traffic observations and access to MITRE ATT&CK knowledge — through prompt-provided description, deterministic MCP retrieval, or agentic MCP tool use — can perform technique-level detection directly, without a dedicated detection model or engineered rule layer of the kind normally used to translate traffic observations into attack classifications. These experiments provide empirical evidence, within the evaluated setting, that qualitative

ATT&CK knowledge accessed through prompts or MCP does not substitute effectively for engineered quantitative traffic features in technique-level flow classification. This finding supports the architectural decision adopted in this thesis to retain deterministic detection upstream of LLM/MCP-based mitigation reasoning, motivated in this section by the absence of prior integrated work and examined empirically in Chapter 7.

3.8. Chapter Summary

This chapter reviewed six areas relevant to the proposed architecture: ML/DL-based intrusion detection, programmable and automated network enforcement, LLM-based security automation, MCP-based agent/tool interaction, 5G-specific ML security, and federated intrusion detection. Existing work provides strong solutions for individual components of the proposed pipeline, including learned detection, programmable enforcement, security orchestration, and LLM-assisted reasoning. However, the literature reviewed for this thesis did not identify an experimentally evaluated architecture combining structured MITRE ATT&CK retrieval, MCP-mediated LLM reasoning, and automated live network enforcement within a single defensive workflow. The use of live enforcement also introduces additional safety and trust requirements beyond those associated with detection-only or advisory systems; these are examined separately in Chapter 8. The next chapter presents the architecture developed to investigate this integration.

4. System Architecture

4.1. Architecture Overview

The proposed system architecturally supports a four-stage pipeline that connects passive traffic analysis to active firewall enforcement through MCP-mediated LLM reasoning. Figure 4.1 illustrates the overall data and control flow.

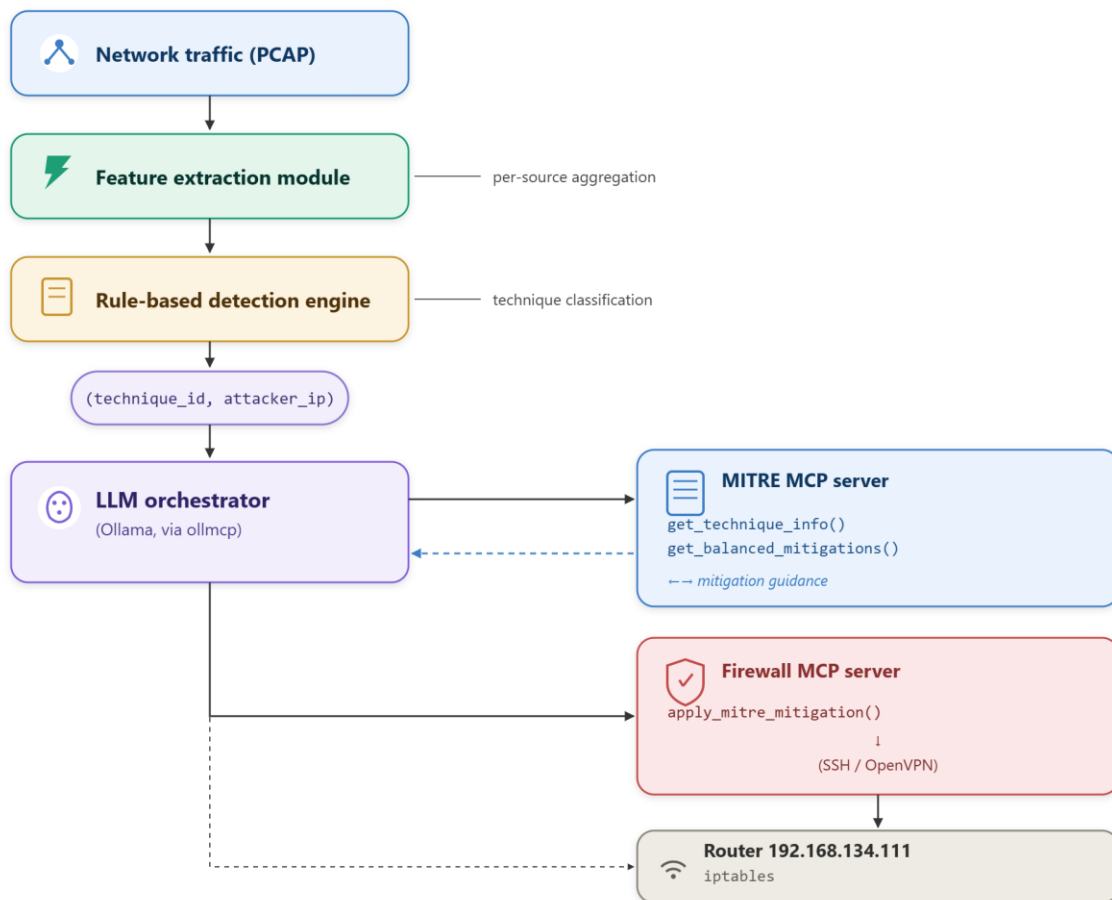


Figure 4.1: Proposed architecture overview. Network traffic is processed by per-source feature extraction and deterministic technique classification, producing a (technique_id, attacker_ip) pair. The orchestration layer exposes both MITRE ATT&CK retrieval and firewall-enforcement capabilities through MCP. The complete conceptual architecture permits MITRE context to inform mitigation selection, while the principal proof-of-concept evaluation uses the constrained execution path described in Section 4.6. *Source: Author’s own diagram.*

The architecture comprises four components, each addressed in the following subsections: the detection and feature extraction module (Section 4.2), the MITRE ATT&CK MCP server (Section 4.3), the firewall enforcement MCP server (Section 4.4), and the LLM orchestrator that coordinates the two servers (Section 4.5). Section 4.6 describes the complete end-to-end pipeline that integrates these components.

This architecture follows the proof-of-concept scope defined in Section 1.1.1 (Scope and Role of AI): all components are implemented and evaluated against synthetic PCAP traffic in a controlled testbed (Chapter 6), with the firewall enforcement applied to a single VPN-connected router rather than a 5G Core UPF. The architectural extensions required for 5G Core deployment are discussed in Section 9.4.

Conceptual and evaluated execution paths. The architecture in Figure 4.1 represents the complete conceptual reasoning path supported by the system, in which the LLM can retrieve ATT&CK technique and mitigation context through the MITRE MCP server before selecting an enforcement action. The principal proof-of-concept evaluation uses a more constrained execution path: the deterministic detector already supplies the ATT&CK technique identifier and attacker address, and the pipeline invokes the predefined `router.apply_mitre_mitigation` enforcement tool through `ollmcp`. MITRE retrieval is exercised separately for stage-level latency measurement and is investigated explicitly in the MCP-assisted experiments of Section 7.7. Consequently, the principal detection-to-enforcement evaluation should not be interpreted as demonstrating that every firewall action was dynamically derived from retrieved MITRE mitigation text.

4.2. Detection and Feature Extraction Module

The detection module operates as a standalone Python process that reads PCAP files, performs per-source feature aggregation, and applies a rule-based classifier to identify traffic matching one of five MITRE ATT&CK techniques.

Feature extraction. For each packet containing an IP layer, the module groups packets by source address and accumulates, for each source: the sequence of packet sizes, packet timestamps, the set of contacted destination IP addresses, and the set of contacted destination ports (extracted from TCP or UDP headers). From these accumulated values, eight per-source features are derived: packet count, average packet size, total bytes, flow duration, number of unique destination IPs, number of unique destination ports, the set of destination ports itself (retained for rule matching), and bytes-per-second throughput. Although these aggregates are referred to as traffic or flow-level features elsewhere in the thesis for convenience, the aggregation unit used by the implementation is the source IP address rather than a conventional bidirectional or 5-tuple network flow.

Detection logic. The classifier applies an ordered sequence of threshold-based rules (Table 4.1), evaluated per source IP. The rule order is significant: more specific rules (e.g., T1048's packet-size threshold) are evaluated before more general rules, preventing a source matching multiple criteria from being misclassified by an earlier, less specific rule. Sources matching none of the five rules are classified as benign and excluded from further processing.

Table 4.1: Detection rules implemented in the rule-based classifier

Order	Technique	Condition	Rationale
1	T1048	$\text{avg_size} > 400$	Synthetic T1048 trace exhibits larger average packet sizes
2	T1568	$\text{unique_dst} \geq 6$	Synthetic T1568 trace contacts many distinct destinations
3	T1498	$\text{pkt_count} < 100$ AND $\text{avg_size} < 130$ AND $\text{unique_dst} \leq 4$ AND $\text{src} \in 198.51.100.0/24$	Synthetic T1498 flood sources originate from the designated external subnet
4	T1190	$200 \leq \text{pkt_count} \leq 300$ AND $\text{avg_size} > 120$ AND $\text{ports} \cap \text{WEB_PORTS} \neq \emptyset$ AND $\text{ports} \cap (\text{REMOTE_PORTS} \setminus \text{WEB_PORTS}) = \emptyset$	Synthetic T1190 trace concentrates on HTTP/HTTPS-adjacent ports
5	T1059	$180 \leq \text{pkt_count} \leq 260$ AND $\text{avg_size} > 100$ AND $\text{ports} \cap \text{REMOTE_PORTS} \neq \emptyset$	Synthetic T1059 trace concentrates on SSH/Kubernetes API ports
—	BENIGN	(no rule matched)	Default classification

In Table 4.1, `ports` denotes the set of distinct destination ports contacted by a source IP, `WEB_PORTS` = {80, 8080, 443, 6443}, `REMOTE_PORTS` = {22, 10250, 2375, 6443}, and `\` denotes set difference (so `REMOTE_PORTS \ WEB_PORTS` = {22, 10250, 2375}).

The derivation of these specific thresholds from the synthetic dataset is described in Section 6.3.

T1498 rule — external validity clarification. The T1498 source-subnet condition ($\text{src} \in 198.51.100.0/24$) is specific to the synthetic dataset and is used to distinguish the externally generated attack population from benign internal hosts (Section 6.2). It is not intended as a general Network DoS detection criterion. An operational implementation would need to replace this address-based discriminator with behavioural, temporal, or topology-aware evidence that does not depend on prior knowledge of the attacker-address range.

T1190 port-set clarification. Port 6443 appears in both the web/API port set (`WEB_PORTS`) and the remote-management port set (`REMOTE_PORTS`) because it is commonly associated with the Kubernetes API. The T1190 exclusion condition therefore removes only the remote-execution-specific ports not already admitted by the web/API set, so that traffic exclusively using port 6443 is not disqualified from matching T1190 by the presence of that same port in the remote-execution set.

Technique identifiers are verified against their official MITRE ATT&CK Enterprise names in Table 2.1 (Section 2.6.3); the correctly identified T1498 (Network Denial of Service) is used consistently throughout the written thesis, except where legacy dataset and code identifiers are explicitly documented in Section 6.2 and the appendices. A note on the underlying dataset filenames, which retain a legacy identifier for practical reasons, is given where the dataset is introduced in Section 6.2.

4.3. MITRE ATT&CK MCP Server

The MITRE MCP server (`mcp_server.py`) is implemented using the FastMCP framework and exposes the local MITRE ATT&CK Enterprise STIX 2.0 bundle (`enterprise-attack.json`) to the LLM orchestrator through three tools.

`get_technique_info(technique_id)` retrieves the named technique object from the STIX bundle via `mitreattack-python`'s `get_object_by_attack_id` method, returning the technique's name and a description truncated to 1000 characters. If the technique ID does not correspond to a known technique, the function returns an explicit "Technique not found" response rather than empty content, reducing ambiguity at the tool boundary and one potential trigger for unsupported LLM inference.

`get_balanced_mitigations(technique_id)` retrieves all course-of-action objects related to the specified technique via `get_mitigations_mitigating_technique`, then applies keyword-based network-relevance filtering to their descriptions — a case-insensitive substring match against the terms `network`, `firewall`, `filter`, `proxy`, `acl`, `allowlist`, `block`, and `rate`, rather than a semantic relevance classification. Up to two matching mitigation excerpts (truncated to 700 characters each) are returned, accompanied by a fixed "Balanced Security Recommendation" block that enumerates general defensive guidance (allowlisting, rate limiting, logging, layered defences) and a reference URL to the official ATT&CK technique page. **This fixed "Balanced Security Recommendation" block is application-specific guidance added by the MCP server and is not part of the MITRE ATT&CK knowledge base.** The server's output therefore combines retrieved ATT&CK content with locally engineered defensive guidance; this distinction is taken into account when interpreting the knowledge-retrieval experiments in Section 7.7. This filtering step addresses the practical observation that MITRE mitigation descriptions span a wide range of organisational and technical controls (e.g., user training, code review), of which only a subset is directly translatable into firewall-level enforcement.

`full_security_analysis(technique_id)` combines the outputs of the two preceding tools into a single structured response, providing the LLM orchestrator with both technique context and mitigation guidance in one tool invocation.

The server is initialised lazily: the STIX bundle is loaded into memory (`get_mitre_data()`) only on the first tool invocation that requires it, and the loaded `MitreAttackData` object is cached for the lifetime of the server process. The server communicates with the MCP client over the stdio transport, consistent with the architecture described in Section 2.5.1.

4.4. Firewall Enforcement MCP Server

The firewall MCP server (`router_firewall.py`) is also implemented using FastMCP and provides the enforcement interface between the LLM orchestrator and a network router accessible via SSH over an OpenVPN tunnel.

Connectivity. The server connects to a fixed router address `<router_vpn_ip>` using the Paramiko SSH library, with an SSH private key (module-level `SSH_KEY_PATH` constant, loaded via `paramiko.RSAKey.from_private_key_file`) rather than a password. Each tool

invocation that requires router interaction establishes a new SSH connection, executes a single command with sudo privileges, captures stdout/stderr, and closes the connection. The `test_connection()` tool provides a diagnostic check, executing a trivial echo command to verify connectivity independent of firewall logic.

Enforcement tools. `block_ip(ip, technique_id)` validates and canonicalises the supplied address using Python’s `ipaddress` module, then constructs and executes the command `iptables -A INPUT -s {ip} -j DROP`, appends the blocked address to an in-memory `blocked_ips` list, and returns a JSON-formatted result including the technique ID associated with the block (if provided) and the exact iptables rule applied; a malformed or injection-bearing address is rejected before any command is constructed. `unblock_ip(ip)` applies the same validation before executing the corresponding `iptables -D` (delete rule) command and removing the address from the in-memory list. `apply_mitre_mitigation(technique_id, attacker_ip)` is functionally equivalent to `block_ip` but is the tool invoked by the detection pipeline (Section 4.6), explicitly associating each enforcement action with the MITRE technique that triggered it. **Despite its name, `apply_mitre_mitigation` does not dynamically translate arbitrary ATT&CK mitigation guidance into firewall configuration.** In the proof-of-concept it implements the predefined mitigation action of blocking the identified source IP, while retaining the triggering ATT&CK technique as contextual metadata. The proof-of-concept still does not implement target allowlisting or enforcement rate limiting, and the SSH/sudo execution path remains broader than would be appropriate for production deployment; these residual limitations are analysed in Chapter 8.

Status tools. `list_blocked()` queries the router’s active iptables INPUT chain directly (`iptables -L INPUT -n | grep DROP`), parsing the output to return the list of currently blocked addresses as observed on the router — rather than relying on the server’s in-memory `blocked_ips` list, which could diverge from actual router state if rules are modified through other means. `get_firewall_stats()` returns the count of active DROP rules alongside connection status metadata.

As discussed in Section 8.4, the current implementation validates and canonicalises IP-address arguments before constructing enforcement commands, preventing malformed or injection-bearing address strings from reaching the command template. However, the proof-of-concept does not implement target allowlisting or enforcement rate limiting, and its SSH/sudo execution path remains broader than would be appropriate for production deployment. These residual limitations are analysed in Chapter 8 and identified as future work in Section 9.6.

4.5. LLM Orchestrator

The orchestration layer is implemented through `ollmcp`, which acts as the MCP host/client and connects an Ollama-hosted LLM inference backend to both MCP servers through the configuration defined in `mcp_router_config.json` (Listing 4.1).

Listing 4.1: MCP server configuration (`mcp_router_config.json`)

This configuration declares both MCP servers under the names `mitre` and `router`, each launched as a subprocess of the configured Python interpreter and communicating with `ollmcp` over `stdio`. The `ollmcp` client exposes the union of both servers' tools to the orchestrating LLM, allowing the model to invoke `mitre.get_technique_info`, `router.apply_mitre_mitigation`, and the remaining tools described in Sections 4.3–4.4 within a single reasoning session.

The orchestrator model is configurable independently of the MCP server implementations. Table 4.2 summarises which model was used at each stage of development and evaluation, since — as discussed below — not every candidate model supports the tool-calling interface this architecture depends on.

Table 4.2: Orchestrator model used at each stage of development and evaluation

Evaluation context	Model	MCP tool calling	Purpose
Early development	<code>gemma3:4b</code>	Development-only; not part of the reported evaluation	Initial integration and testing
Stage 1 direct LLM detection (Section 7.6)	<code>deepseek-r1:32b</code>	No	LLM-only classification (no MCP tools required)
Stage 3 MCP experiments (Section 7.7)	<code>gpt-oss:20b</code>	Yes	Agentic MITRE/MCP evaluation
Pipeline latency evaluation (Section 7.4)	<code>gpt-oss:20b</code>	Yes	MCP/LLM-stage latency measurement

This model-agnostic property follows directly from the MCP architecture (Section 2.5.2): because tool capabilities are described via JSON Schema and discovered at runtime, neither MCP server implementation depends on the specific LLM connected to it. This model-agnosticism holds on the server side without qualification; however, Section 7.7.1 reports an empirically discovered client-side limitation relevant to this claim: not every Ollama-hosted model supports tool-calling at the API level — `deepseek-r1:32b`, used for Stage 1 (Table 4.2), returns an explicit error when tools are supplied, requiring the substitution of the tool-capable `gpt-oss:20b` for the MCP-agentic experiments reported in Section 7.7. The MCP servers themselves remain unmodified across this substitution; the limitation is a property of the connecting LLM's API surface, not of the MCP architecture.

4.6. End-to-End Detection-to-Enforcement Pipeline

The complete pipeline (`evaluate_mcp_pipeline.py`) integrates the components described in Sections 4.2–4.5 as follows:

1. Feature extraction (Section 4.2) is applied to a PCAP file, producing one feature vector per observed source IP address.
2. Detection (Table 4.1) classifies each source IP as either BENIGN or as matching one of the five MITRE techniques.
3. For each non-benign classification, the pipeline invokes `call_firewall("router.apply_mitre_mitigation", {"technique_id": ...,`

"attacker_ip": ...}) via ollmcp, which routes the request through the LLM orchestrator (Section 4.5) to the firewall MCP server (Section 4.4). (`call_firewall` is a historical function name retained from early development; despite the name, it is a generic MCP tool-invocation helper used for calls to either MCP server, not a firewall-specific function.)

4. For latency measurement (Section 6.4.3), an additional call to `mitre.get_technique_info` (Section 4.3) precedes the enforcement call, allowing the MITRE lookup and rule deployment stages to be timed separately.
5. Enforcement results in an iptables DROP rule being applied on the router `<router_vpn_ip>` for the identified attacker IP address.

Figure 4.2 illustrates the complete MCP-enabled reasoning path supported by the architecture, including ATT&CK technique and mitigation retrieval before enforcement; the principal deterministic evaluation path used for the results in Chapter 7 follows the constrained workflow described in steps 1–3 and 5 above, omitting the MITRE-retrieval step (step 4) except where explicitly measuring MITRE-stage latency.

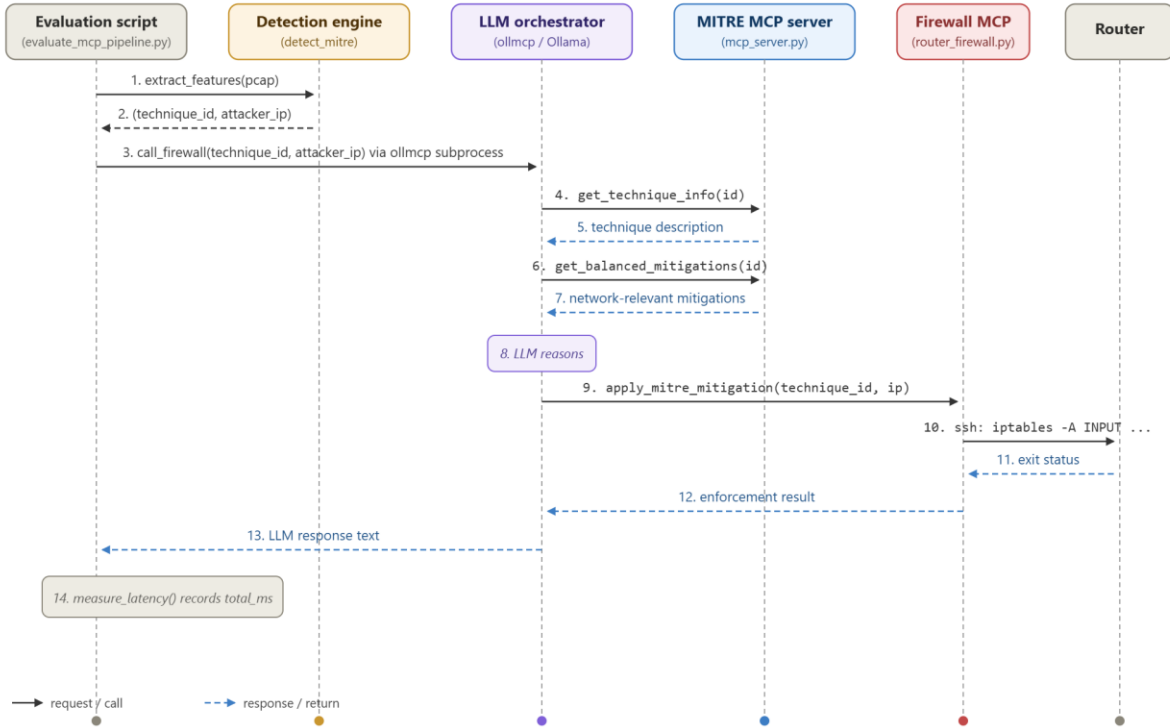


Figure 4.2: Sequence diagram of the complete MCP-enabled reasoning path supported by the architecture. The diagram includes ATT&CK technique and mitigation retrieval before enforcement. The principal deterministic evaluation path uses the constrained workflow described in Section 4.6, while the full MITRE-assisted path is evaluated separately in Section 7.7. *Source: Author’s own diagram.*

The architecture supports the conceptual flow defined in Objective 3 (Section 1.4): detect an attack technique → retrieve relevant ATT&CK context → select a mitigation action →

invoke the firewall enforcement tool. The principal proof-of-concept evaluation uses a constrained version of this flow in which the deterministic detector supplies the technique identifier and the predefined blocking action is invoked directly through MCP. MITRE retrieval and agentic use of ATT&CK context are evaluated separately in the experiments reported in Section 7.7. The empirical evaluation of both execution paths is presented in Chapters 6 and 7.

4.7. Chapter Summary

This chapter presented the architecture of the proposed detection-to-enforcement system. The proof-of-concept separates deterministic traffic aggregation and technique classification, ATT&CK knowledge retrieval, MCP-based firewall enforcement, and LLM-mediated tool orchestration into modular components. The architecture supports a complete ATT&CK-informed reasoning path, while the principal evaluation uses a constrained deterministic-to-enforcement path and evaluates MCP-assisted ATT&CK reasoning separately. Chapter 5 describes the implementation of these components and the software environment used for evaluation.

5. Implementation

5.1. Development Environment

The system was implemented in Python (version verified as 3.14.x in the evaluation environment) within a virtual environment (.venv), running on a Windows host accessed through a WSL (Windows Subsystem for Linux) path structure, within a project directory referred to below as `<project_root>/`. Table 5.1 summarises the software components, their roles, and their versions where verified.

Table 5.1: Development environment components

Component	Library / Tool	Version	Role
Python runtime	CPython	3.14.4	Executes evaluation/MCP code
MCP server framework	mcp (Anthropic MCP SDK), via mcp.server.fastmcp.FastMCP	1.27.2	Implements both MCP servers
MITRE ATT&CK access	mitreattack-python (MitreAttackData)	6.1.0	Parses STIX 2.0 bundle
SSH client	paramiko	5.0.0	Remote command execution on router
PCAP processing	scapy (rdpcap)	2.7.0	Reads synthetic traffic captures
Data handling	pandas	3.0.3	Ground-truth loading, statistics
Numerical computing	numpy	2.4.6	Underlying array operations (pandas/scipy dependency)
Statistical functions	scipy.stats	1.17.1	Confidence interval computation
MCP client	ollmcp (via uvx)	0.35.0	Connects LLM to MCP servers
LLM runtime	Ollama	0.13.0 (remote inference server, <code><ollama_host></code>)	Hosts gemma3:4b / deepseek-r1:32b / gpt-oss:20b
Enforcement transport	OpenVPN Connect / OpenSSH	3.8.0 (build 4528) / 10.2p1 (Ubuntu-2ubuntu3.6), OpenSSL 3.5.5	Connects to router at <code><router_vpn_ip></code>

Note on the Ollama version. The version reported above (0.13.0) was verified directly on the remote inference server (`<ollama_host>:11434`) that hosts the LLM inference workload for all reported results, via the `--host` argument passed to `ollmcp` (Section 4.5). This differs from the Ollama installation present on the local Windows development host used for other tooling (version 0.34.0), which plays no role in the reported experiments; this local/remote version discrepancy is noted here to avoid ambiguity, since only the

server-side version (0.13.0) is relevant to reproducing the results in Chapter 7. This server-side version is notably older than the Ollama releases current at the time of writing (stable releases in the 0.22–0.30 range as of early-to-mid 2026); the inference server was not updated during the evaluation period reported in this thesis, and this version should be treated as a fixed characteristic of the reported environment rather than a representative or currently recommended Ollama version for future replication attempts. The direction and magnitude of this version’s effect on the reported results cannot be determined retrospectively; however, it is plausible to distinguish two categories of finding by their likely sensitivity to this factor. Findings tied to a model’s inherent training (e.g., deepseek-r1:32b’s lack of tool-calling support, Section 7.7.1) are unlikely to be affected by the Ollama runtime version, since tool-calling support is a property of how the model itself was trained and packaged. Findings tied to runtime performance characteristics (e.g., the latency variability discussed in Section 7.4, and general inference throughput) are more plausibly sensitive to the runtime version, since subsequent Ollama releases have specifically targeted performance, caching, and tool-calling-pipeline improvements. This distinction is noted as a limitation in Section 9.3 and motivates re-running the latency measurements on an updated runtime as a direction for future work (Section 9.6).

Note on the MCP framework version. The `mcp` package version reported above (1.27.2) was confirmed via `pip show mcp` after activating the correct project virtual environment; an earlier check in a misconfigured environment had failed to locate this package, which briefly raised a false concern about an unpinned dependency. The two MCP servers import `FastMCP` from `mcp.server.fastmcp`, provided by this package.

5.2. MITRE ATT&CK MCP Server Implementation

The MITRE MCP server is implemented in `mcp_server.py`. The server is instantiated as `mcp = FastMCP("MitreSecurityTool")`, and the local STIX bundle path is configured via the `DATA_PATH` constant. In the evaluated environment, `DATA_PATH` is configured as an absolute WSL-accessible path to the local `enterprise-attack.json` file. A production or portable implementation should resolve this location through configuration rather than a hard-coded filesystem path. A minimal `test_tool()` function, returning a fixed confirmation string, is also exposed as an MCP tool to allow connectivity to be verified independently of any MITRE-specific logic (used during the workflow verification described in Section 5.5).

Lazy data loading. The `get_mitre_data()` helper implements a module-level cache: on first invocation, it verifies the existence of `enterprise-attack.json` and raises a `RuntimeError` with an explicit instruction to download the file manually if it is absent; on subsequent invocations, it returns the cached `MitreAttackData` instance without re-parsing the file. This avoids the overhead of re-parsing the (multi-megabyte) STIX bundle on every tool call.

Mitigation extraction. The `get_balanced_mitigations` tool calls `mitre_data.get_mitigations_mitigating_technique(technique.id)`, which returns a list of mitigation-relationship wrapper objects. The helper function `mitigation_object_from_wrapper` normalises these wrappers — handling both dictionary-style (`{"object": ...}`) and attribute-style (`.object`) representations returned by different versions of the underlying library — before extracting each

mitigation's description via `safe_description`. Descriptions are filtered against a fixed keyword list (`["network", "firewall", "filter", "proxy", "acl", "allowlist", "block", "rate"]`) using case-insensitive substring matching; only descriptions containing at least one keyword are retained, up to a maximum of two.

Error handling. All three tools (`get_technique_info`, `get_balanced_mitigations`, `full_security_analysis`) wrap their logic in try/except blocks that catch all exceptions and return a string-formatted error message (e.g., “Error in `get_technique_info`: {str(e)}”) rather than propagating exceptions to the MCP client. This ensures that a malformed technique ID or an unexpected STIX data structure produces an informative tool response rather than terminating the MCP server process. This broad exception handling favours server availability during proof-of-concept evaluation but reduces error specificity. A production implementation should distinguish expected validation/data-access failures from unexpected programming errors and return structured MCP error responses rather than generic strings.

5.3. Firewall Enforcement MCP Server Implementation

The firewall MCP server is implemented in `router_firewall.py` as `app = FastMCP("RouterFirewall")`.

SSH execution helper. The `ssh_to_router(command)` function encapsulates all router communication: it instantiates a `paramiko.SSHClient`, sets `AutoAddPolicy` for host key verification (accepting unknown host keys without prompting — a simplification appropriate for the controlled testbed but noted as a hardening item for production use), and connects to `ROUTER_VPN_IP` on port 22 using key-based authentication — an RSA private key loaded via `paramiko.RSAKey.from_private_key_file(SSH_KEY_PATH)` — rather than a password, with a 10-second connection timeout. It then executes the supplied command prefixed with `sudo`, and returns a (success, output_or_error) tuple. Connections are opened and closed for each invocation rather than pooled.

State tracking. The module-level list `blocked_ips` provides an in-process record of addresses blocked via `block_ip` or `apply_mitre_mitigation` during the current server session. This list is supplementary: as noted in Section 4.4, `list_blocked()` queries the router's actual iptables state directly rather than relying on this list, ensuring that reported blocked addresses reflect ground truth even if the in-memory list becomes stale (e.g., if rules are modified outside the MCP server, or across server restarts).

Command construction. The `ip` argument to `block_ip`, `unblock_ip`, and `apply_mitre_mitigation` is first validated and canonicalised using Python's `ipaddress` module (`ipaddress.ip_address()`), which raises `ValueError` — caught and returned as a structured failure result — for any input that does not parse as a valid IPv4/IPv6 address, before any shell command is constructed. Enforcement commands then follow a fixed template: `iptables -A INPUT -s {ip} -j DROP` for blocking and `iptables -D INPUT -s {ip} -j DROP` for unblocking, with the validated `{ip}` substituted via Python f-string interpolation. `list_blocked()` constructs a pipeline command (`iptables -L INPUT -n | grep DROP | awk '{print $4}'`) executed as a single string via `ssh_to_router`, with the

output parsed by splitting on newlines and filtering empty strings. The final implementation validates and canonicalises the supplied IP address before constructing the enforcement command, rejecting malformed or injection-bearing inputs. The proof-of-concept still does not implement target allowlisting or enforcement rate limiting, and the SSH/sudo execution path remains broader than would be appropriate for production deployment; these residual limitations are analysed in Chapter 8.

5.4. Evaluation Pipeline Implementation

The evaluation pipeline (`evaluate_mcp_pipeline.py`) implements the feature extraction, detection, and statistical evaluation logic described in Sections 4.2 and 6.3.

Feature extraction (`extract_features`). Iterates over packets returned by `scapy.rdpcap`, building a `defaultdict` keyed by source IP address. For each packet with an IP layer, the packet size, timestamp, destination IP, transport protocol, and (for TCP/UDP) destination port are recorded. After processing all packets, per-source aggregate features are computed, including `flow_duration` (computed as the difference between the maximum and minimum observed timestamps, with a floor of 0.001 seconds to avoid division by zero in `bytes_per_sec`).

Detection (`detect_mitre`). Implements the five-rule classifier of Table 4.1 as a sequence of `if/return` statements, with `WEB_PORTS = {80, 8080, 443, 6443}` and `REMOTE_PORTS = {22, 10250, 2375, 6443}` defined as module-level set constants referenced by the T1190 and T1059 rules.

MCP invocation (`call_firewall`). Constructs and executes a subprocess command invoking `uvx ollmcp` with the orchestrator host, model, MCP server configuration file (`mcp_router_config.json`), target tool name, and JSON-serialised tool arguments. Despite its name — a historical artefact of early development, retained in the codebase — this function is a generic MCP tool-invocation helper used for calls to either MCP server, not a firewall-specific function. The subprocess is run with a 60-second timeout and `capture_output=True`; the function returns the subprocess’s `stdout`, or `None` if the subprocess raises an exception. The 60-second timeout is a defensive execution bound chosen to prevent stalled LLM/MCP subprocesses from blocking the evaluation indefinitely; it is not treated as a performance target.

Latency measurement (`measure_latency`). For a given technique ID and attacker IP, separately times (a) a `mitre.get_technique_info` call and (b) a `router.apply_mitre_mitigation` call, using `time.perf_counter()` before and after each `call_firewall` invocation, recording two named intervals — referred to in this thesis as **MITRE-stage latency** (variable name `mcp_query_ms` in the code) and **enforcement-stage latency** (variable name `rule_deploy_ms` in the code) — and their sum, the **combined measured pipeline latency** (`total_ms`). These measurements represent coarse end-to-end stage timings rather than isolated tool-execution times. In particular, the measured intervals include subprocess launch, `ollmcp`/LLM processing, and MCP dispatch in both stages, and additionally include SSH connection establishment in the enforcement stage, in addition to the underlying ATT&CK lookup or `iptables` operation itself.

Statistical computation (compute_statistics). Computes mean, sample standard deviation (`ddof=1`), and a 95% confidence interval via `scipy.stats.t.interval`. A guard clause returns `(mean, mean)` as the confidence interval when `std == 0` or `n < 2`, avoiding the division-by-zero that otherwise produces NaN values (Section 6.6 discusses the interpretation of this case).

Baseline (baseline_static_iptables). Implements a single-rule baseline classifier — `predicted = {f["src"] for f in features if f["packet_count"] > 100}` — representing a naive static threshold on packet count, used for comparison in Section 7.1. Despite the function's name (a legacy artefact of early development), this baseline is more precisely described throughout this thesis as the static packet-count threshold baseline, to avoid implying a comparison against a full iptables-based firewall product.

5.5. End-to-End Workflow Verification

Functional integration of the two MCP servers with the LLM orchestrator was verified prior to the full evaluation runs (Chapter 7) using the `test_tool()` and `test_connection()` diagnostic tools exposed by the MITRE and firewall MCP servers respectively. `test_tool()` returns a static string confirming that the MITRE MCP server process is reachable and responding to tool invocations via `ollmcp`. `test_connection()` performs a live SSH round-trip to the router over the OpenVPN tunnel, confirming both VPN connectivity and SSH credential validity independent of any firewall rule modification. These diagnostic tools allowed connectivity issues (Section 7.5 notes observed `ollmcp` invocation errors during early evaluation runs) to be isolated from detection-logic issues during development. Both diagnostics completed successfully in the final testbed configuration before the evaluation runs, confirming MCP-server reachability and live SSH connectivity to the enforcement router.

5.6. Chapter Summary

This chapter described the implementation of the deterministic traffic-analysis pipeline, MITRE ATT&CK MCP server, firewall enforcement MCP server, `ollmcp` orchestration environment, and SSH-based iptables backend. The implementation preserves modular separation between detection, threat-intelligence access, reasoning, and enforcement. IP-address arguments at the enforcement boundary are validated and canonicalised before command construction; remaining proof-of-concept limitations include dataset-calibrated detection rules, coarse stage-level latency instrumentation, per-invocation SSH connections, broad exception handling, the absence of target allowlisting and enforcement rate limiting, and an SSH/sudo execution model that requires further least-privilege hardening for production use. Chapter 6 defines the experimental methodology used to evaluate these components.

6. Experimental Methodology

This chapter describes the methodology used to evaluate the proposed architecture (Chapter 4) and its implementation (Chapter 5). Section 6.1 describes the testbed configuration. Section 6.2 describes the synthetic dataset. Section 6.3 describes the derivation of the detection rules used by the rule-based classifier (Table 4.1). Section 6.4 defines the performance metrics and the latency measurement procedure. Section 6.5 describes the sequential input-size scaling experiment design. Section 6.6 discusses the repeated-execution methodology and its statistical interpretation.

6.1. Testbed Configuration

The evaluation testbed consists of a single Windows host running a WSL2 (Windows Subsystem for Linux) environment, in which the detection pipeline, the two MCP servers (Sections 5.2, 5.3), and the evaluation script (Section 5.4) execute as Python processes within a shared virtual environment (.venv). The LLM orchestrator (Section 4.5) is hosted remotely on an Ollama server (<ollama_host>:11434) and accessed over the network by the ollmcp client.

The firewall enforcement node is a separate router, reachable from the testbed host via an OpenVPN tunnel at the address <router_vpn_ip> (Section 5.3). All enforcement actions (Section 4.4) are applied to this single node via SSH; no 5G Core network function, UPF, or SBA interface is present in the testbed. This is consistent with the proof-of-concept scope established in Section 4.1 (and stated explicitly, as a bulleted list, in Section 1.1): the testbed evaluates the detection-to-enforcement pipeline in isolation from any 5G Core deployment, with the architectural extensions required for 5G Core integration discussed separately in Section 9.4.

Three Ollama-hosted models were available on the evaluation server: gemma3:4b, deepseek-r1:32b, and gpt-oss:20b. Because the MCP architecture decouples tool definitions from the orchestrating model (Section 2.5.2), the pipeline can in principle be re-run against any of the three models by editing a single configuration field, without modifying either MCP server implementation. In practice, the specific model used varies across the results reported in Chapter 7, reflecting both deliberate experimental design and a practical constraint discovered during evaluation:

- The core detection-to-enforcement pipeline results (Sections 7.1–7.5), including the latency measurements in Section 7.4, were obtained using **gpt-oss:20b**, as configured in mcp_router_config.json (Listing 4.1).
- The LLM-based detection investigation without the rule-based filter (Section 7.6) evaluated multiple models directly as part of its experimental design; the best-performing and final selected configuration (Section 7.6.3) uses **deepseek-r1:32b**, while an earlier baseline configuration (Section 7.6.1) uses gpt-oss:20b for comparison.
- The MCP-retrieval experiments (Section 7.7) uniformly use **gpt-oss:20b**, not by preference but by necessity: Section 7.7.1 reports that deepseek-r1:32b — used for the Section 7.6 best result — does not support Ollama’s tool-calling interface

(returning an explicit “does not support tools” error), making it unusable for any experiment requiring agentic MCP tool invocation. This limitation is a property of the connecting LLM’s API surface rather than of the MCP servers themselves (Section 4.5).

This model-agnostic property of the MCP architecture is discussed further in Section 9.6 as a basis for future comparative latency evaluation across models; the tool-calling support limitation discovered in Section 7.7.1 is noted there as a practical constraint on which models can serve as the orchestrator for agentic (as opposed to prompt-only) LLM configurations.

6.2. Synthetic Dataset

The evaluation dataset consists of five pairs of PCAP and CSV files, each corresponding to one of the five MITRE ATT&CK techniques selected in Section 2.6.3 (T1190, T1048, T1059, T1498, T1568). Each PCAP file contains synthetically generated packet captures representing a mixture of benign and attacker traffic; each corresponding CSV file provides ground-truth labels (label column, values attack or benign) keyed by source IP address (src_ip).

Note on the T1498 dataset files. The physical dataset files, the internal mitre_attack label column values, and the source code’s internal handling of this technique (Sections 5.4, Appendix A/C) all retain the identifier T1495 (e.g., T1495_network_dos.pcap), reflecting an initial technique misidentification made prior to the correction documented in Section 2.6.3. Regenerating these files under the corrected identifier was outside the scope of this correction; the internal T1495 identifier is therefore preserved throughout the codebase and dataset artifacts purely as a naming convention with no bearing on the correctness of the reported results, while all technique identification in the written thesis, tables, and MCP technique-lookup queries uses the correct identifier, T1498. The legacy dataset and code identifiers for the network-DoS scenario remain T1495 for artifact compatibility. Before constructing any live MITRE MCP query in the affected Stage 3 experiments, an explicit mapping step (T1495 → T1498) was therefore introduced (Appendix C.6). Experiments 3E and 3G were rerun using the corrected T1498 identifier, and the corrected results are reported in Section 7.7. Experiment 3H performs no live MITRE MCP query and therefore required no identifier patch; it was rerun unchanged as an independent repeatability check. The underlying dataset files and internal labels remain unchanged.

Table 6.1 summarises the per-source aggregate feature values observed for the attacker population in each dataset, computed by the extract_features function (Section 5.4) and used as the basis for the detection rule derivation described in Section 6.3.

Table 6.1: Summary of attacker traffic characteristics per technique

Technique	Approx. Packet Count	Approx. Avg. Packet Size (bytes)	Distinctive Ports	Unique Destinations	Number of Attacker IPs
T1190	~260	~131	80, 8080, 6443	3	small (single-digit)
T1048	~280	~653	varied	4	small (single-digit)
T1059	~220	~118	22, 10250, 6443	3	small (single-digit)
T1498	~45–75	~105	—	≤4	70
T1568	~579	~75	—	8	small (single-digit)

The T1498 (Network DoS) dataset is structurally distinct from the other four: it represents a distributed attack involving 70 distinct attacker source IP addresses, each contributing a relatively small number of packets (45–75) from the 198.51.100.0/24 subnet, whereas the remaining four datasets each represent a small number of attacker sources generating a larger per-source packet count. This distinction directly motivates the sequential input-size scaling experiment design described in Section 6.5, which is conducted exclusively on the T1498 dataset.

Benign traffic in each dataset is generated from a fixed set of nine source IP addresses exhibiting relatively repetitive traffic patterns (Section 7.2.2). As discussed in Section 7.2.2 (Construct Validity), this benign population is substantially smaller and less diverse than would be expected in an operational network, and represents a limitation of the dataset’s representativeness rather than of the detection logic itself.

6.3. Detection Rule Derivation

The detection rules implemented in `detect_mitre` (Table 4.1) were derived through an iterative, empirical process applied directly to the synthetic dataset described in Section 6.2. This process proceeded as follows.

First, the `extract_features` function (Section 5.4) was applied to each PCAP file, and the resulting per-source feature vectors were inspected separately for the attacker sources (identified via the ground-truth CSV) and the benign sources. This inspection produced the aggregate characteristics summarised in Table 6.1.

Second, an initial rule set was constructed using only the two most readily distinguishing features — packet count and average packet size — under the hypothesis that these alone might separate attacker traffic from benign traffic across all five techniques. This initial rule set was found to produce false positives: specifically, benign DNS traffic (destination port 53) and benign internal service traffic exhibited packet-count ranges that overlapped with the packet-count range associated with T1498 attacker traffic, causing these benign sources to be misclassified as T1498 (Section 7.5).

Third, the rule set was iteratively refined by incorporating additional discriminating features for each technique: destination port-set membership (used to distinguish T1190

and T1059, which target web-facing and remote-execution ports respectively, Table 4.1), the number of unique destination IP addresses (used both for T1568, which is characterised by contact with a large number of distinct destinations, and — at the opposite end of the range — for T1498, whose attacker sources are constrained to at most 4 unique destinations, Table 6.1), and source-subnet membership (also used for T1498, restricting matches to the 198.51.100.0/24 range associated with the synthetic distributed attack). The average-packet-size threshold for T1048 ($\text{avg_size} > 400$) was set based on the substantially larger mean packet size (~ 653 bytes) observed for T1048 attacker traffic relative to all other categories (Table 6.1), reflecting the bulk-transfer characteristic of the exfiltration technique.

Finally, the order of rule evaluation (Table 4.1, “Order” column) was fixed so that more specific rules are evaluated before more general ones — for example, the T1048 size-based rule is evaluated first because a sufficiently large average packet size is a strong and largely unambiguous indicator, whereas the T1059 and T1190 rules, which rely on overlapping port-set and packet-count ranges, are evaluated last and are mutually exclusive by construction (the T1190 rule explicitly excludes sources whose ports also fall in the remote-execution set outside the web-port set).

This derivation process — refining thresholds against the same dataset subsequently used for evaluation — constitutes the primary internal validity threat discussed in Section 7.2.1, and is acknowledged explicitly here as the origin of that threat rather than left implicit.

6.4. Performance Metrics and Latency Measurement

6.4.1. Detection Performance Metrics

For each technique, detection performance is evaluated using standard Precision, Recall, and F1 score, computed by the `evaluate` function (Section 5.4) as:

- $\text{Precision} = \frac{TP}{TP+FP}$
- $\text{Recall} = \frac{TP}{TP+FN}$
- $F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

where TP, FP, and FN are computed by comparing the set of source IP addresses classified as the target technique (predicted, produced by `detect_mitre`) against the ground-truth set of attacker IP addresses (truth, loaded from the CSV by `load_attackers`). A guard against division by zero returns 0.0 for any ratio whose denominator is zero.

6.4.2. Static Packet-Count Threshold Baseline

To contextualise the detection performance of the proposed rule-based classifier, a static packet-count threshold baseline (function name `baseline_static_iptables`, retained as a legacy identifier) is evaluated alongside it. The baseline implements a single, technique-agnostic rule: a source IP is classified as an attacker if and only if its packet count exceeds

100 (`packet_count > 100`). This baseline represents a simple, technique-agnostic traffic-classification comparator inspired by static volumetric filtering — the kind of coarse threshold consistent with the traditional firewall architectures discussed in Section 2.3 — rather than an evaluation of any specific firewall product or rule, and serves as a reference point against which the benefit of technique-specific, multi-feature detection rules (Table 4.1) can be assessed (Section 7.1). This baseline is deliberately simple: it is intended to represent a technique-agnostic static-rule comparator illustrating the selectivity gained by technique-specific detection rules, not to represent a state-of-the-art commercial NGFW or IDS product. The comparison in Section 7.1 should accordingly be read as evaluating the value of technique-specific rule derivation over undifferentiated volumetric filtering, not as a claim of superiority over modern, actively maintained security products.

6.4.3. Pipeline Latency Measurement

End-to-end pipeline latency is measured by the `measure_latency` function (Section 5.4), which times two stages of the MCP-mediated pipeline separately for a given technique identifier and attacker IP address:

1. **MITRE-stage latency** (variable name `mcp_query_ms` in the code): the elapsed wall-clock time for a `mitre.get_technique_info` tool call, invoked via `call_firewall` (Section 5.4), which internally constructs and executes an `ollmcp` subprocess invocation against the configured LLM orchestrator (Section 4.5).
2. **Enforcement-stage latency** (variable name `rule_deploy_ms` in the code): the elapsed wall-clock time for a `router.apply_mitre_mitigation` tool call, invoked identically.

The two measurements are summed to produce the **combined measured pipeline latency** (`total_ms`), the figure reported in Table 7.3. These labels describe coarse end-to-end stage timings rather than isolated tool-execution times: each measurement encompasses the full round trip through the `ollmcp` client, the LLM orchestrator’s reasoning and tool-selection process, the corresponding MCP server’s tool execution (including, for the enforcement stage, the SSH round trip to the enforcement router described in Section 5.3), and the return of the result to the evaluation script — not solely the underlying ATT&CK lookup or iptables operation itself. Latency measurement is performed once per technique (on the first of the 30 repeated runs described in Section 6.6), reflecting the practical cost of invoking the `ollmcp` subprocess, which is substantially higher than the cost of the local detection logic (Section 7.4). Because this design takes a single measurement per technique rather than repeated samples, the resulting figures are subject to whatever transient server and network conditions prevailed at measurement time; Section 7.4 reports a specific instance of this sensitivity, observed during a later diagnostic re-measurement.

6.5. Sequential Input-Set Scaling Experiment Design

The scalability experiment, implemented by `run_scalability` (Section 5.4), is conducted exclusively on the T1498 dataset, which is the only dataset containing a sufficiently large

attacker population (70 source IP addresses, Section 6.2) to support a meaningful attacker-set-size sweep (Section 7.3).

For each value of n from 1 to 10, the experiment selects the n attacker sources with the highest packet count (via descending sort on `packet_count`), applies the rule-based detection logic (`detect_mitre`) to this sample of n sources, and computes Precision, Recall, and F1 by comparing the resulting predicted set against the full 70-IP ground-truth set (truth), together with the wall-clock time required to classify the n -source sample (`latency_ms`). Because each n -source sample is formed by selecting the attacker sources with the highest packet counts rather than by random sampling, the experiment characterises the scaling behaviour of a high-volume subset of the T1498 attacker population. It should not be interpreted as an unbiased estimate of performance across arbitrary n -source subsets.

A critical methodological point, central to the interpretation of the results in Section 7.3, is that the denominator of the **global attacker coverage** calculation ($TP + FN$) is fixed at the size of the full ground-truth set (70) for every value of n , regardless of how many attackers are included in the sample. Consequently, even when every attacker in a sample of size n is correctly detected (yielding $TP = n$, $FN = 0$ within the sample), the reported global attacker coverage is $n/70$, not 1.0. This design choice — retained from the original implementation — produces the counter-intuitive coverage values discussed in Section 7.3.1 (ranging from 0.0143 at $n=1$ to 0.1429 at $n=10$) and is explicitly not a measure of per-sample detection completeness. The primary per-sample completeness metric used in Section 7.3.1 is instead the sample detection rate ($\text{Detected}/n$, i.e., the proportion of the sampled n attackers actually flagged), which is 1.000 at all tested n ; Precision (also 1.000 at all n) separately indicates the absence of false positives within each sample, and the two metrics are discussed together in Section 7.3.1.

6.6. Repeated Execution and Statistical Considerations

To verify the operational consistency of the implemented detection pipeline, each evaluation scenario (`run_pipeline`, Section 5.4) was executed 30 times ($\text{NUM_RUNS} = 30$). It is important to clarify the purpose and interpretation of this repetition.

The synthetic dataset described in Section 6.2 consists of static PCAP and CSV files. The feature extraction procedure (`extract_features`) and the rule-based detection logic (`detect_mitre`) are both deterministic functions of their input: given the same PCAP file, `extract_features` produces identical per-source feature vectors on every invocation, and `detect_mitre` applies fixed threshold comparisons that yield identical classifications for identical feature vectors. Consequently, repeated execution of the pipeline against the same dataset produces identical Precision, Recall, and F1 values on every run, with zero variance across all 30 executions for all five evaluated techniques — as confirmed by the `std`: 0.0 values recorded for every technique in `results_v3.json` (Table 7.1).

Repeated execution was therefore performed to verify implementation consistency — confirming that the pipeline behaves reproducibly and contains no sources of non-determinism such as race conditions in feature extraction or order-dependent set

operations — rather than to support statistical inference. Because the resulting standard deviation is zero by construction, the 95% confidence intervals computed by `compute_statistics` collapse to single points (mean, mean) for all proposed-system metrics and provide no inferential information beyond the point estimate itself.

This is a deliberate departure from the conventional use of repeated trials in machine learning evaluation, where repetition typically captures variance arising from stochastic elements such as random initialisation, data shuffling, or sampling. No such stochastic elements are present in the detection component of the pipeline: the dataset is fixed, and the detection rules contain no random components. As a result, claims of statistical significance based on these 30 repetitions would not be methodologically valid, and no such claims are made in this thesis (Section 7.2.4).

A meaningful statistical evaluation — incorporating, for example, multiple independently generated traffic samples, randomised benign-traffic injection, or cross-validation across distinct synthetic datasets — would require a dataset generation methodology that introduces genuine inter-trial variance. This is identified as a direction for future work in Section 9.6.

It is noted, separately, that the latency measurements (`measure_latency`, Section 6.4.3) are not deterministic in the same sense, since they depend on the LLM orchestrator’s inference time and on network/SSH round-trip times to the enforcement router. However, latency was measured only once per technique ($n=1$ in `results_v3.json`, Section 7.4) rather than across all 30 runs, and the latency confidence intervals reported in Table 7.3 (where $n=1$) should accordingly be read as single-point measurements rather than as statistically characterised distributions. Repeated latency sampling ($n > 1$) across multiple independent invocations is identified as a further direction for future work in Section 9.6.

This deterministic/non-deterministic distinction drawn here for the rule-based pipeline stands in direct contrast to the LLM-based detection configuration evaluated in Section 7.6.5, where the same repeated-execution methodology (10 runs, rather than 30, reflecting the substantially higher per-run cost of LLM inference) is applied for the opposite reason: because LLM inference is potentially non-deterministic even under a fixed configuration, and the resulting statistics (mean, standard deviation, minimum, maximum) are reported precisely because they are expected to — and, prior to isolating an infrastructure confound, did — exhibit genuine inter-run variance, unlike the zero-variance result obtained here for the deterministic rule-based detector.

7. Results and Discussion

This chapter presents the experimental evaluation of the proposed system in three parts, summarised in **Figure 7.0**: the rule-based detection engine is evaluated on its own (Sections 7.1–7.5), followed by two complementary investigations into LLM-based detection conducted without the rule filter — an LLM-only configuration (Section 7.6) and an MCP-assisted configuration (Section 7.7) — before the findings from all three are synthesised into the architectural decision that motivates the remainder of the thesis (Section 7.8).

7.1. Detection Performance

Table 7.1 presents the detection performance of the proposed system across all five evaluated MITRE ATT&CK techniques, alongside the static packet-count threshold baseline (Section 6.4.2), based on the methodology described in Chapter 6.

Table 7.1: Detection performance by technique (proposed system vs. static baseline)

Technique	Precision	Recall	F1	Baseline TP/FP/FN	Baseline Precision	Baseline Recall	Baseline F1
T1190	1.000	1.000	1.000	1 / 2 / 0	0.333	1.000	0.500
T1048	1.000	1.000	1.000	1 / 2 / 0	0.333	1.000	0.500
T1059	1.000	1.000	1.000	1 / 2 / 0	0.333	1.000	0.500
T1498	1.000	1.000	1.000	0 / 0 / 70	0.000	0.000	0.000
T1568	1.000	1.000	1.000	1 / 2 / 0	0.333	1.000	0.500

Standard deviation across the 30 repeated executions described in Section 6.6 is 0.0 for all techniques and metrics; confidence intervals are not reported in this table, as they collapse to point estimates and provide no inferential value (Section 6.6).

The proposed system achieves perfect classification (Precision = Recall = F1 = 1.0) on all five techniques, substantially outperforming the static baseline, which achieves F1 = 0.5 on four techniques and F1 = 0.0 on T1498. The baseline’s failure on T1498 arises because its single rule (`packet_count > 100`) does not match any of the 70 T1498 attacker sources, each of which has a packet count between approximately 45 and 75 (Table 6.1) — illustrating the limitation of static, single-threshold rules discussed in Section 2.3.3. Conversely, the baseline’s non-zero false positives (FP = 2) on the other four techniques reflect benign sources whose packet count exceeds 100 but which are correctly excluded by the proposed system’s additional discriminating features (Table 4.1).

These results represent the calibrated performance of the proposed detector on the evaluated synthetic dataset and should not be interpreted as estimates of real-world detection performance. These perfect scores must, however, be interpreted with substantial caution. Sections 7.1.1 and 7.1.2 provide two further empirical checks that contextualise this result, before Section 7.2 addresses the remaining threats to validity.

7.1.1. Cross-Dataset Validation

Section 7.2.1 identifies rule-derivation circularity as the primary internal validity threat to the results in Table 7.1: the detection rules (Table 4.1) were derived by inspecting the same

dataset subsequently used for evaluation. While this threat cannot be fully eliminated without independently generated traffic (Section 9.6), one aspect of it can be partially addressed using the existing five datasets, and is reported here.

Each of the five synthetic traces (Section 6.2) contains traffic from the same fixed pool of nine benign source IP addresses (10.96.0.11, 10.96.0.13–10.96.0.16, 10.168.2.75, 192.168.134.68, 192.168.134.192, 192.168.134.195), corresponding to core network function services (NRF, AMF, UDM, AUSF, SMF, UPF/UE-side hosts, and DNS resolvers) in the underlying free5GC/Kubernetes topology. Critically, the specific benign traffic generated for each of the nine hosts differs across the five traces — for example, host 10.96.0.16 generates 135 packets in the T1190 trace, 132 in the T1048 trace, 124 in the T1059 trace, 99 in the T1498 trace, and 141 in the T1568 trace, reflecting independent 10-minute capture sessions (Section 6.2, README manifest) rather than a single replayed dataset.

This structure permits a cross-dataset check: for each of the five techniques, the corresponding detection rule (Table 4.1) was applied not only to the trace from which it was derived, but to the benign-host traffic of all five traces combined (45 benign-host observations: 9 hosts $\times$ 5 independently captured sessions). A false positive in this check would indicate that a rule, while correctly calibrated to its own trace, misclassifies benign traffic captured under slightly different conditions — a weaker but meaningful generalisation signal that does not require new data collection.

Result: Across all 45 benign-host observations and all five detection rules, zero false positives were produced (Table 7.1b). Each rule correctly classified all 45 benign-host observations as BENIGN, and each technique’s rule correctly and exclusively identified its own attacker host(s) (Table 7.1). Table 7.1a summarises this same result organised by source dataset (9 benign hosts checked per dataset, evaluated against that dataset’s own detection rule) rather than by rule, for a more direct at-a-glance view.

Table 7.1a: Cross-dataset validation, by source dataset

Dataset	Benign Hosts Checked	False Positives
T1190	9	0
T1048	9	0
T1059	9	0
T1498	9	0
T1568	9	0

Table 7.1b: Cross-dataset benign-host false positive check, by detection rule (applied across all 45 combined benign-host observations)

Technique rule	Benign-host observations checked	False positives
T1048 (avg_size > 400)	45	0
T1568 (unique_dst >= 6)	45	0
T1498 (pkt<100, avg_size<130, subnet 198.51.100.0/24)	45	0
T1190 (200≤pkt≤300, avg_size>120, web ports)	45	0
T1059 (180≤pkt≤260, avg_size>100, remote ports)	45	0

This result strengthens, but does not eliminate, the internal validity threat discussed in Section 7.2.1. It demonstrates that the detection rules are not narrowly overfit to the exact packet-level realisation of benign traffic in a single 10-minute capture — the benign traffic varies in packet count by up to 17% across the five independent captures of the same nine hosts (e.g., host 10.96.0.16: 99–141 packets), and the rules remain correct. However, this check does not address the remaining components of the internal validity threat: all five attacker traces were generated by the same synthetic traffic generator using the same underlying parameters for each technique (Section 6.2), and no benign traffic from hosts outside the fixed nine-host pool, nor any attacker traffic generated under different parameters, was evaluated. The construct and external validity threats discussed in Sections 7.2.2 and 7.2.3 — concerning the representativeness of both the synthetic attacker traffic and the limited nine-host benign population relative to a production network — remain fully applicable.

It is also worth noting an additional observation from this analysis relevant to Section 7.2.3 (External Validity): inspection of the T1498 attacker population confirms the README manifest’s reported attack/benign packet split (4,200 attack packets across 70 sources, 302 benign packets), and that all 70 T1498 attacker sources share an identical destination-port signature (`{80, 2152, 8000}`) and an identical unique-destination count (4), despite per-source packet counts varying from 45 to 75 and average packet sizes varying from approximately 96 to 122 bytes. This indicates that the synthetic T1498 traffic generator introduced volumetric variation between attacker sources while holding flow-level structural features (ports, destination count) constant — a property that benefits the current rule (which depends only on the structurally constant features plus the subnet condition) but that would not necessarily hold for a real distributed denial-of-service attack, in which port and destination-count diversity across attacker sources is more likely (Section 7.2.2).

7.1.2. Robustness Under Feature Perturbation

Sections 7.1 and 7.1.1 report Precision = Recall = F1 = 1.000 for all five techniques, both on each technique’s own dataset and across the combined 45-observation benign-host pool. As discussed in Section 7.2.1, these perfect scores reflect rules that were calibrated against the exact feature values present in the synthetic dataset, and a natural question is how sensitive this perfect performance is to small deviations in those feature values — deviations of the kind that would be expected if the same attack techniques were executed with slightly different timing, payload sizes, or packet rates.

To investigate this, a controlled perturbation experiment was conducted. For each of the five datasets, the two continuous features used most heavily by the detection rules (Table 4.1) — `packet_count` and `avg_size` — were each independently multiplied by a factor $1 + U(-p, p)$, where $U(-p, p)$ is a uniform random variable and p is a perturbation level ranging from 0% to 30%. All other features (destination ports, unique destination count, source subnet) were left unchanged, since these are categorical/set-valued and a multiplicative perturbation model does not apply to them directly. For each perturbation level $p > 0$, 100 independent trials were run with different random seeds, and the mean and standard deviation of F1 across the 100 per-trial values were computed (an initial pilot run using 10

trials produced a materially less stable estimate for one technique, discussed below, motivating the larger trial count adopted here). Table 7.1c reports the results.

Table 7.1c: Detection performance under feature perturbation (packet_count and avg_size, $\pm p\%$), mean F1 $\pm$ SD across 100 trials

Technique	p=0% (1 trial)	p=5%	p=10%	p=20%	p=30%
T1190 (F1)	1.000	1.000 $\pm$ 0.000	0.900 $\pm$ 0.302	0.620 $\pm$ 0.488	0.430 $\pm$ 0.498
T1048 (F1)	1.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000
T1059 (F1)	1.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	0.850 $\pm$ 0.359	0.480 $\pm$ 0.502
T1498 (F1)	1.000	1.000 $\pm$ 0.000	0.999 $\pm$ 0.002	0.976 $\pm$ 0.014	0.927 $\pm$ 0.024
T1568 (F1)	1.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000

Precision remained 1.000 for all techniques at all perturbation levels — across all five rules and all tested perturbation magnitudes, not a single false positive was introduced by perturbing benign or other-technique traffic into a target technique’s feature range. All degradation in F1 is attributable entirely to false negatives (Recall): perturbed attacker traffic occasionally fell outside the rule’s threshold range and was classified as BENIGN.

Only two of the five rules — T1048 and T1568 — show no degradation at all, even under $\pm 30\%$ perturbation. T1048’s threshold (`avg_size > 400`) has a very large margin relative to the observed attacker value (~ 653 bytes, Table 6.1), so even a -30% perturbation (to ~ 457 bytes) remains well above the threshold. T1568’s threshold (`unique_dst >= 6`) operates on a feature not perturbed in this experiment.

An initial pilot run of this experiment (10 trials per perturbation level, rather than 100) had suggested that T1190 also showed no degradation, alongside T1048 and T1568. Increasing the trial count to 100 revealed this to be an artifact of insufficient sample size: because the T1190 dataset contains a single attacker source, each individual trial’s F1 is a binary outcome (0 or 1) rather than a graded value, and with only 10 trials, chance alone can produce a run in which no trial happens to cross the rule’s threshold. The large standard deviations observed for T1190 (e.g., 0.488 at $p=20\%$) reflect exactly this binary, high-variance structure, and are a direct consequence of the technique’s small attacker population (Table 6.1) rather than a data-quality issue with the experiment itself. This finding is noted as a methodological lesson: perturbation or robustness experiments involving a small number of positive instances require a correspondingly larger number of trials before a “no degradation” conclusion can be drawn with confidence.

The remaining three rules — T1190, T1059, and T1498 — degrade measurably under larger perturbations, though for different reasons and at different rates:

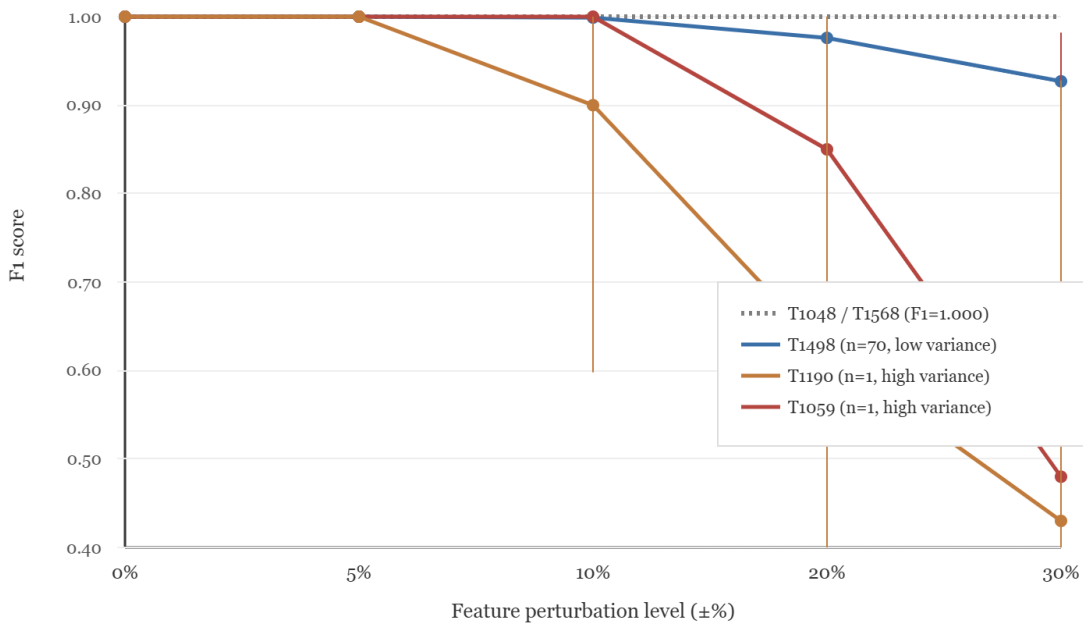
T1190 ($200 \leq \text{pkt_count} \leq 300$, `avg_size > 120`, web ports) degrades from F1 = 1.000 at $p \leq 5\%$ to a mean F1 of 0.900 at $p = 10\%$, 0.620 at $p = 20\%$, and 0.430 at $p = 30\%$. Contrary to the packet-count-margin explanation initially hypothesised for this rule, the weaker constraint is the `avg_size` threshold: the single T1190 attacker’s baseline average packet size (~ 131 bytes, Table 6.1) exceeds the rule’s threshold (`avg_size > 120`) by a margin of only about 8%, so even a 10% perturbation has a substantial probability of pushing `avg_size` below 120 and causing the source to fall through to BENIGN — considerably

narrower margin than the packet-count range (200–300 around an observed value of ~260) suggested in isolation.

T1059 ($180 \leq \text{pkt_count} \leq 260$, $\text{avg_size} > 100$, remote ports) degrades from $F1 = 1.000$ at $p \leq 10\%$ to a mean $F1$ of 0.850 at $p = 20\%$ and 0.480 at $p = 30\%$. The single T1059 attacker source has a baseline packet count of 220 (Table 6.1) — close to the midpoint of its 180–260 range, but the range itself spans only $\pm 18\%$ of the midpoint, meaning a $+20\%$ perturbation ($220 \rightarrow 264$) can push the packet count above the upper bound of 260, causing the source to fall through to BENIGN.

T1498 ($\text{pkt_count} < 100$, $\text{avg_size} < 130$, subnet match) degrades most gradually, from $F1 = 1.000$ at $p \leq 5\%$ to a mean $F1$ of 0.999 at $p = 10\%$, 0.976 at $p = 20\%$, and 0.927 at $p = 30\%$, with correspondingly small standard deviations (≤ 0.024) reflecting the much larger attacker population (70 sources) for this technique, which averages out the same binary per-source outcome over many more instances. Since the baseline T1498 attacker avg_size values cluster in the 96–122 range (Table 6.1) with the threshold at 130, a margin of only 6–25%, a $+20\%$ to $+30\%$ perturbation can push avg_size above 130 for sources already close to the boundary.

Figure 7.1 (corrected) — F1 Under Feature Perturbation, Mean $\pm$ SD (N=100 trials)



Error bars show ± 1 SD across 100 trials (clipped at plot boundaries where SD is large relative to the mean).

Precision remained 1.000 for all techniques at all levels (Section 7.1.2).

Figure 7.1: F1 score under feature perturbation, by technique. T1048 and T1568 retain $F1 = 1.000$ across all perturbation levels; T1498 degrades gradually and with low variance from $p=10\%$ onward, reflecting its large (70-source) attacker population; T1190 and T1059 degrade more sharply and with high variance, reflecting their single-attacker populations and correspondingly binary per-trial outcomes.

Comparison with the unperturbed and cross-dataset results (Sections 7.1, 7.1.1)

This experiment provides a different and complementary perspective from the cross-dataset check in Section 7.1.1. The cross-dataset check (45 independent benign-host observations across five separately captured 10-minute traces) found zero exceptions to the perfect-score results, suggesting the rules generalise across independent captures of the same underlying traffic-generation process. The perturbation experiment in this section, by contrast, asks how the rules behave under synthetic deviations from that process — and finds that for three of the five rules (T1190, T1059, and T1498), performance is not invariant: F1 drops measurably once feature values are perturbed by 10-20% or more.

Taken together, these two results support a more precise statement than “the system achieves perfect detection” (Section 7.1): the system achieves perfect detection on traffic whose flow-level features fall within the same ranges as the synthetic traffic used for rule derivation (Sections 6.2, 6.3), and this includes independently captured traffic from the same generation process (Section 7.1.1), but detection degrades for three of five techniques (T1190, T1059, T1498) once feature values drift sufficiently far from the values observed during rule derivation. T1048 has sufficient threshold margin relative to its observed attacker feature values to absorb $\pm 30\%$ perturbation without degradation; T1568 remains unchanged under the applied perturbation model because its discriminative feature (`unique_dst`) was held constant rather than perturbed, and its perfect result should not be read as evidence of genuine margin in the way T1048’s is. The remaining three rules (T1190, T1059, T1498) do not retain perfect F1 under perturbation, because their thresholds were derived to sit close to the observed attacker feature values with limited margin (Section 6.3).

This finding refines, rather than overturns, the conclusions of Section 7.1: it does not indicate that the detection rules are incorrect, but it does provide direct, quantitative evidence — requested implicitly by the internal validity discussion in Section 7.2.1 — of how much margin exists in each rule, and identifies T1190, T1059, and T1498 as the three techniques whose detection rules would most benefit from threshold widening or replacement with a learned (rather than fixed-threshold) classifier (Section 9.6) prior to any deployment beyond the evaluated dataset. Notably, Precision remained perfect (1.000) at all perturbation levels for all techniques: the observed degradation is entirely a Recall (false-negative) effect, and the rules did not become more prone to false alarms under perturbation — only less sensitive.

7.2. Threats to Validity

The evaluation presented in Section 7.1 is subject to several threats to validity that must be explicitly acknowledged in order to correctly interpret the reported results. Following the classification proposed by Wohlin et al. [1], four categories of validity threats are considered.

7.2.1. Internal Validity

The most significant internal validity threat in this work is rule derivation bias. As documented in Section 6.3, the detection rules applied in the evaluation pipeline were derived through empirical analysis of the same dataset used for testing. Threshold values for packet count, average packet size, destination port sets, unique-destination counts, and source-subnet membership were selected by inspecting the ground-truth attacker traffic in each PCAP file, and were iteratively refined to eliminate observed false positives (Section 6.3). As a consequence, the rules are optimally calibrated for the evaluation dataset by construction, and the Precision = 1.0, Recall = 1.0, F1 = 1.0 scores reported in Table 7.1 reflect this calibration rather than demonstrated generalizable detection capability.

This constitutes a form of circular validation and must be interpreted accordingly. The reported metrics demonstrate that the implemented pipeline correctly identifies the patterns present in the synthetic dataset; they do not demonstrate that the same rules would achieve comparable performance on independently generated or real-world traffic. Sections 7.1.1 and 7.1.2 provide two empirical checks that partially address this threat — cross-dataset validation across independently captured benign traffic (zero false positives across 45 observations), and a feature-perturbation stress test that quantifies the margin of each rule and identifies three techniques (T1190, T1059, T1498) whose Recall degrades under perturbations of 10-20% or more. Together, these checks demonstrate that the perfect scores in Table 7.1 are not an artifact of trivial overfitting to a single capture, but they do not eliminate the underlying circularity: all rules were still derived from, and all attacker traffic was still generated by, the same synthetic process.

A second internal validity threat is the deterministic nature of the repeated evaluation runs, discussed in Section 6.6: because the dataset and detection logic are both deterministic, the 30 repeated executions do not constitute independent trials in the statistical sense.

7.2.2. Construct Validity

The primary construct validity threat concerns the representativeness of the synthetic dataset relative to real attack traffic. While the dataset provides labelled ground truth and controlled attack conditions, it may not capture the full diversity of real-world attack traffic. In particular: real T1190 traffic would involve application-layer exploit payloads and post-exploitation communication, neither of which is present in the synthetic dataset, which represents this technique purely through aggregate flow statistics (packet count, size, destination ports); real T1498 traffic in a 5G environment would involve GTP-U flooding at the N3 interface (Section 2.4.2), which differs structurally from the generic UDP-style flood represented in the dataset — and, as noted in Section 7.1.1, the synthetic T1498 attacker population holds destination ports and unique-destination count constant across all 70 sources, which may not hold for a real distributed attack; and benign traffic is limited to nine source IPs with relatively simple, repetitive traffic patterns (Section 6.2), whereas production network traffic exhibits far greater diversity.

A related construct validity limitation concerns the perturbation experiment itself (Section 7.1.2): the perturbation operates at the level of derived features (`packet_count` and

avg_size), independently multiplying each by uniform random noise, rather than by regenerating or replaying packet-level traces. Consequently, a perturbed feature combination is not guaranteed to correspond to a physically realizable traffic capture — for example, packet_count and total_bytes are perturbed independently of one another despite being related quantities in real traffic, and destination ports and unique-destination counts are held fixed regardless of how packet_count and avg_size are perturbed. The perturbation experiment should therefore be understood as a synthetic stress test of the detection rules’ threshold margins under independent feature-level noise, rather than as a simulation of how a real adversary might vary their traffic to evade detection.

These gaps mean that the detection rules, while effective on the synthetic dataset and robust to moderate perturbation for three of five techniques (Section 7.1.2), have not been validated against realistic traffic distributions or against traffic generated by actual exploit tooling.

A further, more serious construct validity threat was identified after the Stage 3 experiments (Section 7.7) were first run, concerning the T1495/T1498 technique identifier correction discussed in Section 2.6.3. The configurations that query the real MITRE MCP server for the network-DoS candidate — 3A, 3E, and 3G — initially did so using the internal identifier “T1495” rather than the officially correct “T1498” (Section 2.6.3). (Experiments 3B/3F made no MCP calls and were therefore unaffected by the identifier issue.) To resolve this construct-validity threat, Experiments 3E and 3G were rerun with an explicit identifier-mapping patch applied immediately before each live MITRE MCP query, ensuring that the MITRE server was queried using T1498 rather than the legacy T1495 dataset label. Experiment 3H issues no live MITRE MCP query and therefore required no mapping patch; it was rerun unchanged as a separate repeatability check. The results reported in Sections 7.7.4–7.7.7 reflect these corrected and repeated executions.

The corrected rerun confirms, rather than overturns, the qualitative-versus-quantitative finding: for both 3E and 3G, classification quality using the genuinely correct T1498 (Network Denial of Service) description was essentially unchanged from, or marginally lower than, the originally reported figures using the mistaken T1495 (Firmware Corruption) content (Section 7.7.4, 7.7.5). This indicates that the observed degradation was not an artefact of retrieving irrelevant content for one technique, but reflects a genuine property of qualitative, general-purpose MITRE ATT&CK knowledge relative to the dataset-calibrated, quantitative descriptions used in Experiment 3D. Experiment 3H’s compressed-MITRE content required no correction, since manual inspection confirmed its hard-coded description had already captured the correct Network Denial of Service semantics; a repeated run of the unmodified 3H script nonetheless produced a materially different Macro-F1 (0.40 versus the originally reported 0.20, Section 7.7.6) with identical code and model, which is reported and discussed as evidence of genuine run-to-run variance in this experimental configuration, distinct from the identifier-correction issue affecting 3E and 3G.

7.2.3. External Validity

The generalizability of the results is limited by four factors. First, as established in Section 6.1, the evaluation environment contains no 5G Core component; performance claims apply only to the evaluated testbed and cannot be directly extrapolated to 5G Core deployments. Second, the T1498 detection rule relies on source IP prefix matching against the 198.51.100.0/24 range (Table 4.1) — a rule that is specific to the subnet structure of this dataset and would not generalise to attackers operating from arbitrary subnets. Third, only five of the more than 200 techniques documented in MITRE ATT&CK Enterprise (Section 2.6.1) are evaluated; whether the rule-based approach can be extended to a broader technique set without substantial manual re-engineering per technique remains an open question. Fourth, the `SIMILAR_NEIGHBOR_COUNT` relational feature used in the best Stage 1 LLM configuration (Section 7.6.3) restricts cluster candidacy to public/external IP address ranges, a dataset-specific heuristic that reflects this testbed’s particular benign-traffic pattern and must not be interpreted as a general security assumption — a real coordinated attack originating from compromised internal CNFs, pods, MEC workloads, or subscriber-side hosts would not be flagged by this heuristic. The perturbation results of Section 7.1.2 add a further dimension to the first three of these threats: even within the five evaluated techniques, the T1190, T1059, and T1498 rules’ narrow thresholds mean their external validity to traffic with even moderately different volumetric characteristics is lower than that of T1048 and T1568.

7.2.4. Conclusion Validity

As discussed in Section 6.6, the absence of statistical variance across the 30 evaluation runs means that standard statistical significance tests (e.g., Mann-Whitney U, Wilcoxon signed-rank) cannot be meaningfully applied to the results in Table 7.1. Comparisons between the proposed system and the static baseline are therefore based on point estimates rather than statistically significant differences. The perturbation experiment (Section 7.1.2), by contrast, does introduce genuine inter-trial variance (via random feature perturbation) and supports the comparisons reported in Table 7.1c on that basis. Additionally, the scalability evaluation (Section 7.3) exhibits a result — Global attacker coverage remaining below 0.15 across all tested attacker-set sizes — that, without explanation, would appear to indicate a conclusion-validity problem; Section 7.3 addresses this directly.

Table 7.1d: Summary of threats to validity

Threat	Impact	Mitigating Evidence / Response
Internal (Section 7.2.1)	High	Cross-dataset validation (Section 7.1.1); feature-perturbation stress test (Section 7.1.2)
Construct (Section 7.2.2)	Medium	Acknowledged gap between synthetic and real attack traffic; noted as future work (Section 9.6)
External (Section 7.2.3)	High	Scope limited to evaluated testbed and five techniques; 5G applicability analysed separately (Section 9.4)
Conclusion (Section 7.2.4)	Low	Repeatability analysis (Section 7.6.5); perturbation trials provide genuine inter-trial variance where claimed

7.3. Sequential Input-Set Scaling with Attacker-Set Size

This experiment evaluates computational scaling with input-set size, not true runtime concurrency. No parallel traffic arrival, concurrent packet processing, concurrent MCP calls, queueing, or enforcement-load contention is simulated; the detection logic is applied sequentially to a sample of n pre-extracted feature vectors, and the wall-clock time to classify that sample is recorded. The results below should therefore be read as characterising how classification cost and per-sample accuracy scale as the size of the input attacker set grows from 1 to 10, not as a demonstration of the pipeline’s behaviour under genuinely concurrent load.

Table 7.2 presents the results of this experiment, described in Section 6.5, conducted on the T1498 dataset (70 attacker IPs). For comparison, Table 7.2 also reports the corresponding per-sample classification latency for the T1190, T1048, T1059, and T1568 datasets at the same attacker-set sizes, all of which achieve Precision = Sample Recall = F1 = 1.000 at every value of n (since each of these datasets contains only a small, single-digit number of attacker IPs, all of which fall within the $n=1..10$ sampling range).

Table 7.2: Detection scaling results for T1498 (attacker-set size $n = 1-10$)

Attackers (n)	Detected / n (Sample Detection Rate)	Precision	Latency (ms)	Global attacker coverage (n/70, legacy metric — see note below)
1	1/1 (100%)	1.000	0.004	0.0143
2	2/2 (100%)	1.000	0.003	0.0286
3	3/3 (100%)	1.000	0.001	0.0429
4	4/4 (100%)	1.000	0.001	0.0571
5	5/5 (100%)	1.000	0.002	0.0714
6	6/6 (100%)	1.000	0.002	0.0857
7	7/7 (100%)	1.000	0.004	0.1000
8	8/8 (100%)	1.000	0.005	0.1143
9	9/9 (100%)	1.000	0.004	0.1286
10	10/10 (100%)	1.000	0.008	0.1429

The primary metrics for this experiment are **Sample Detection Rate** (Detected / n , the fraction of attackers in the current sample that were correctly detected — 100% at every value of n) and **per-sample classification latency**, both of which directly answer the question this experiment is designed to ask: does detection accuracy and computational cost hold up as the attacker-set size grows? The **Global attacker coverage** column ($n/70$, computed against the full 70-attacker ground truth, Section 6.5) is retained only as a legacy artifact of the underlying evaluation script’s implementation, which always evaluates the sample’s predictions against the complete ground-truth set rather than against a truth set scoped to the sample; it produces the small, monotonically increasing values shown, which do not indicate degrading detection performance and should not be used to assess this

experiment's outcome. F1 computed from Global attacker coverage is omitted from Table 7.2 for the same reason and is not a meaningful summary statistic for this experiment.

Table 7.2b: Per-sample classification latency (ms) for the remaining four techniques, attacker-set size n=1-10

Attackers (n)	T1190	T1048	T1059	T1568
1	0.006	0.003	0.005	0.004
2	0.004	0.002	0.003	0.002
3	0.002	0.001	0.002	0.001
4	0.001	0.000	0.002	0.001
5	0.001	0.000	0.002	0.000
6	0.001	0.000	0.001	0.000
7	0.001	0.001	0.002	0.001
8	0.001	0.001	0.085	0.100
9	0.001	0.000	0.002	0.003
10	0.001	0.000	0.001	0.001

For T1190, T1048, T1059, and T1568, Precision = Sample Detection Rate = 1.000 at every tested attacker-set size (these datasets contain only a small number of attacker IPs, all within the n=1..10 range, so all are detected regardless of n). The two latency outliers visible in Table 7.2b (T1059 at n=8: 0.085 ms; T1568 at n=8: 0.100 ms) are consistent with normal measurement jitter from the host operating system scheduler during a sub-millisecond timing measurement, and do not indicate a systematic scaling effect — values immediately before and after n=8 return to the sub-0.01 ms range for both techniques.

7.3.1. Discussion of Detection Scaling Results

The Global attacker coverage values in Table 7.2 — ranging from 0.0143 at n=1 to 0.1429 at n=10 — may, at first glance, appear to indicate poor detection performance, with the system “missing” the majority of attackers as the attacker-set size increases. This interpretation would be incorrect, and the result requires explanation rooted in the experimental design described in Section 6.5 — which is precisely why Global attacker coverage is demoted to a secondary, legacy metric in Table 7.2 above, in favour of Sample Detection Rate.

As specified in Section 6.5, Global attacker coverage at each value of n is computed as $TP / (TP + FN)$, where the denominator (TP + FN) equals the size of the complete ground-truth set — all 70 T1498 attacker IPs — regardless of how many attackers are included in the current sample of size n. Consequently, when the sample contains n attackers and all n are correctly detected (TP = n, FN = 0 within the sample), the resulting Global attacker coverage is n/70, yielding $1/70 \approx 0.0143$ for n=1 and $10/70 \approx 0.1429$ for n=10 — precisely the values observed in Table 7.2. Sample Detection Rate (Detected / n), by contrast, is 100% at every value of n and directly reflects the pipeline's actual per-attacker detection accuracy within the tested sample.

The Precision value of 1.000 across all values of n confirms that every attacker included in the sample is correctly detected with zero false positives. The detection rule for T1498 (Table 4.1) — based on packet count, average packet size, and source subnet membership — produces identical per-sample accuracy regardless of the attacker-set size processed.

The correct interpretation is: the detection logic exhibits perfect per-attacker accuracy at all tested attacker-set sizes ($n=1$ to 10), and the pipeline’s per-sample latency remains stable — sub-0.01 ms in nearly all cases across all five techniques (Tables 7.2 and 7.2b) — as n increases, indicating no observable computational scaling bottleneck within this range under sequential processing. As emphasised above, this finding characterises scaling with input-set size under sequential classification, not the pipeline’s behaviour under genuinely concurrent traffic arrival or concurrent MCP/enforcement calls, which was not evaluated (Section 9.3).

This finding does not, however, demonstrate that the full population of 70 simultaneous attackers can be processed with equivalent performance, since the largest tested sample ($n=10$) represents only 14.3% of the total T1498 attacker population. Because each n -source sample is formed by selecting the attacker sources with the highest packet counts rather than by random sampling (Section 6.5), the experiment characterises the scaling behaviour of a high-volume subset of the T1498 attacker population; it should not be interpreted as an unbiased estimate of performance across arbitrary n -source subsets. Evaluating the complete 70-attacker scenario under an unbiased sampling scheme, and assessing whether detection or enforcement latency degrades non-linearly at that scale — for instance, due to the sequential nature of the `apply_mitre_mitigation` MCP calls in the pipeline (Section 5.4), each of which currently incurs the multi-second LLM latency discussed in Section 7.4 — remains an open question for future work (Section 9.6).

A methodologically preferable design for future scalability evaluation would define Recall relative to the sample’s own ground truth (i.e., the n attackers included in that specific test) rather than the full 70-attacker population. This would more directly convey per-sample detection accuracy and avoid the counter-intuitive presentation in Table 7.2. This refinement is noted as a methodological limitation of the present evaluation.

7.4. Pipeline Latency Analysis

Section 6.4.3 defines pipeline latency as the sum of two measured stages: MITRE-stage latency and enforcement-stage latency, both invoked through the LLM orchestrator (Section 4.5). Table 7.3 presents the measured total latency for all five evaluated techniques, as recorded in `results_v3.json`.

Table 7.3: Measured pipeline latency (gpt-oss:20b orchestrator)

Technique	Total Latency (ms)	n
T1190	2,323.38	1
T1048	1,501.81	1
T1059	1,597.87	1
T1498	1,440.60	1
T1568	1,361.49	1

Note: each value is a single ($n = 1$) snapshot measurement from the original run and should be read as a representative, order-of-magnitude observation rather than a stable estimate. A

later diagnostic re-measurement of the T1498 figure differed materially (4,944.41 ms versus the 1,440.60 ms reported here); see Appendix B.3 (Tables B.7 and B.8).

Across all five techniques, the measured total pipeline latency ranges from approximately 1.36 to 2.32 seconds, with a mean of approximately 1,645 ms. This is substantially lower than the 2.4–7.8 second range observed in an earlier evaluation pass using a different orchestrator model (recorded separately during development), and is consistent with gpt-oss:20b being a comparatively smaller and faster model than deepseek-r1:32b. The latency figures in Table 7.3 should therefore be read as specific to the gpt-oss:20b configuration described in Section 6.1; a systematic comparison of latency across the three available orchestrator models (gemma3:4b, deepseek-r1:32b, gpt-oss:20b) was not performed and is identified as future work (Section 9.6).

A note on latency reproducibility. A diagnostic re-measurement was subsequently performed to verify the T1498 figure in Table 7.3, isolating a single MCP query and rule-deployment call under conditions intended to match the original measurement. This re-measurement, performed at a later date, returned a substantially higher latency (approximately 4.6–4.9 seconds, depending on configuration) than the originally recorded 1,440.60 ms, despite targeting the same model and the same single-call measurement scope. This discrepancy is attributed to the latency measurement’s sensitivity to transient conditions outside the pipeline’s own control — including Ollama server load, whether the target model was already resident in memory (“warm”) versus freshly loaded (“cold”) at measurement time, and network conditions to the remote Ollama host (Section 6.1) — rather than to any change in the pipeline’s logic. This observation is consistent with the general finding of this section, that the combined MCP/LLM path dominates measured latency (Section 7.4.1): a measurement so heavily dependent on this orchestrated path will necessarily inherit its own run-to-run variability, a factor not fully captured by a single-measurement-per-technique design (Section 6.4.3). The absolute latency values in Table 7.3 should therefore be interpreted as representative of the order of magnitude of pipeline latency under the conditions prevailing at measurement time, rather than as a precise, reproducible constant; repeated latency sampling under controlled, monitored server conditions is identified as a refinement for future work (Section 9.6).

Unlike the earlier evaluation pass, no ollmcp invocation errors were recorded for any of the five techniques in this run: latency was successfully measured for all five MITRE techniques, in contrast to only two of five in the previous pass (Section 7.5 discusses the error categories observed in that earlier pass for completeness, since they remain relevant to the operational reliability discussion in Section 7.6).

7.4.1. Latency Decomposition and Bottleneck Identification

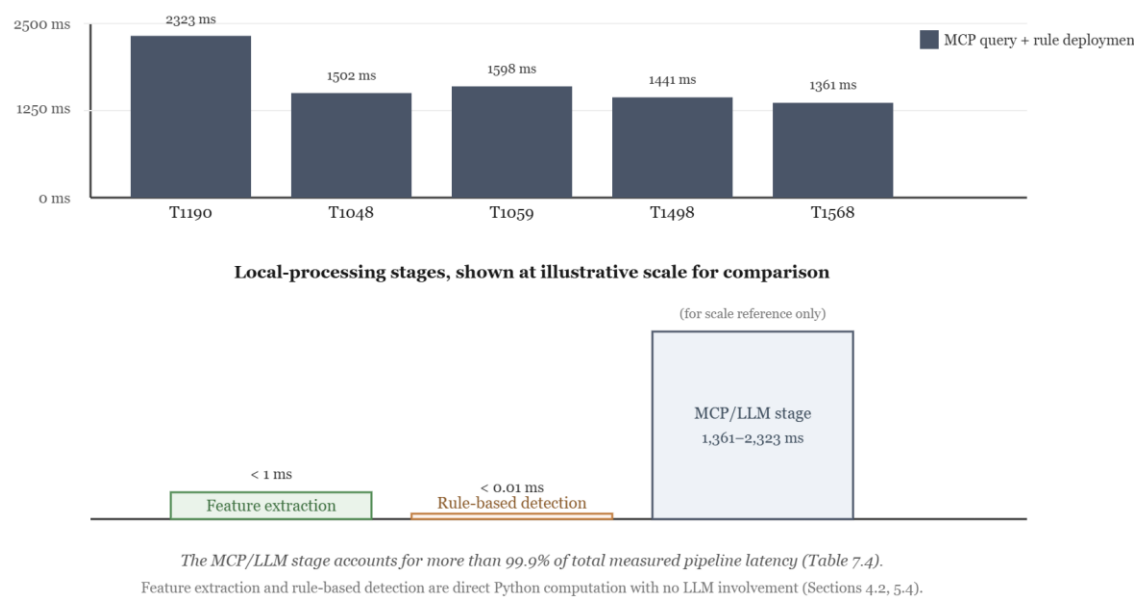
Table 7.3 reports only the total latency (MITRE query + rule deployment combined), as recorded by measure_latency (Section 6.4.3). To identify the dominant contributor to this total, it is informative to compare these values against the latencies of the pipeline’s other stages, measured independently:

Table 7.4: Approximate latency decomposition by pipeline stage

Stage	Approximate Latency	Source
Feature extraction (per PCAP)	< 1 ms	detection_time_ms, Table 7.1 evaluation runs
Rule-based detection (per source)	< 0.01 ms	per-sample latency, Tables 7.2 and 7.2b
MCP query + rule deployment (combined)	1,361–2,323 ms	Table 7.3

Figure 7.4.1 — Pipeline Latency Decomposition by Stage and Technique

Top: total MCP/LLM latency per technique (linear scale). Bottom: local-processing stages (illustrative, not to scale).



The combined MCP/LLM invocation path accounts for more than 99.9% of the measured pipeline time relative to local feature extraction and classification, for all five techniques (**Figure 7.4.1**). Feature extraction and rule-based detection — the components implemented as direct Python computation without LLM involvement (Sections 4.2, 5.4) — complete in under one millisecond, consistent with the sub-0.01 ms per-sample latencies observed in the sequential input-size scaling experiment (Tables 7.2 and 7.2b, which do not invoke the MCP/LLM stage).

The current instrumentation does not separately measure model inference, subprocess startup, MCP dispatch, SSH connection establishment, and tool execution. The measurements therefore identify the orchestrated MCP/LLM path as the dominant bottleneck, while the exact contribution of the individual subcomponents remains to be quantified; instrumenting each of these stages separately is identified as a concrete improvement for future work (Section 9.6).

This finding has direct implications for the applicability claims discussed in Section 1.4 (RQ4). The ITU-defined latency budget for URLLC user-plane traffic is on the order of 1 millisecond [5], while the eMBB latency budget is less commonly specified as a hard bound but is typically discussed in the tens-of-milliseconds range for interactive applications. The measured end-to-end pipeline latency of 1.36–2.32 seconds exceeds both of these budgets by three orders of magnitude. The current implementation is therefore not suitable for direct in-line mitigation of URLLC-class traffic, and would introduce substantial response delay even relative to eMBB latency expectations. This is an important and explicit limitation of the proof-of-concept evaluated in this thesis (Section 1.2), and the specific architectural changes that would be required to address it — such as removing the LLM from the latency-critical path entirely, discussed below — are identified as priorities for future work (Section 9.6).

7.5. Failure Cases and Error Analysis

This section documents operational failure modes encountered during the development and evaluation of the proposed system, consistent with the expectation that an MSc evaluation chapter document not only successful outcomes but also failure modes encountered during experimentation. Failures are separated into those encountered during development of the detection rules (Section 7.5.1) and those encountered during the evaluation runs reported in Chapter 7 (Section 7.5.2), since these two categories have different implications for interpreting the results in Sections 7.1–7.4.

7.5.1. Development-Time Failures

Detection rule iteration failures. As documented in Section 6.3, an initial version of the detection rules — using only packet-count and average-packet-size thresholds without port-set, unique-destination, or subnet conditions — produced false positives on benign DNS traffic (port 53) and internal service traffic, which were classified as T1498 due to overlapping packet-count ranges. This failure was encountered and resolved during rule derivation, before any of the results reported in Sections 7.1–7.5 were produced. It is included here, rather than omitted as a solved problem, because it directly informed the final rule set (Table 4.1) and illustrates the rule-derivation process underlying the internal validity threat discussed in Section 7.2.1.

7.5.2. Evaluation-Time Failures

MCP invocation errors. During an earlier evaluation pass using the deepseek-r1:32b orchestrator, ollmcp invocations during latency measurement returned a command-line usage error (Usage: ollmcp [OPTIONS] ... Try 'ollmcp -h' for help) for three of the five techniques (T1190, T1059, T1498), rather than executing the requested tool call. This error indicates a malformed command-line invocation of the ollmcp client (Section 5.4) — for example, an incompatibility between the argument format expected by the installed ollmcp version and the arguments constructed by `call_firewall`. Because `call_firewall` returns `None` on subprocess failure (Section 5.4) rather than raising an exception, these errors did not interrupt the evaluation pipeline; however, they prevented latency measurement for the affected techniques in that pass. In the evaluation pass reported in

Table 7.3 (using gpt-oss:20b), this error did not recur for any of the five techniques, suggesting the issue may have been related to the specific orchestrator configuration or ollmcp invocation arguments used in the earlier pass rather than a structural defect in the pipeline; however, the underlying cause was not isolated, and the intermittent nature of this failure mode is noted as an operational reliability concern in Section 7.6.

This failure mode highlights a gap between the evaluation pipeline’s detection metrics (Table 7.1), which depend only on the local feature extraction and rule classification logic and are therefore unaffected by MCP connectivity issues, and the pipeline’s enforcement capability, which depends on successful MCP/LLM invocation. The Precision/Recall/F1 = 1.0 results (Table 7.1) should therefore be understood as characterising the detection component in isolation; they do not, by themselves, constitute evidence that the enforcement component executes reliably across all orchestrator configurations.

LLM hallucination. No instances of LLM hallucination (fabricated technique IDs or non-existent tool names) were observed during evaluation with either orchestrator configuration. However, because `measure_latency` exercises only two specific tool calls (`mitre.get_technique_info` and `router.apply_mitre_mitigation`) per technique, this observation is limited to those two tool-invocation pathways and does not constitute a comprehensive characterisation of the orchestrator’s tool-selection behaviour across the full tool surface described in Section 2.5.2.

7.6. LLM-Based Detection Without Rule Filtering

Experimental question for this section. With the deterministic rule layer of Section 7.1 removed entirely, can an LLM classify each source IP’s MITRE ATT&CK technique directly from traffic features alone? The variable isolated across Sections 7.6.1–7.6.4 is the *input representation* given to the model (individual-host vs. batched vs. relational/population context); success is judged primarily by Macro-F1 and per-technique F1 across all five techniques, not by Micro-F1 alone, since the dataset’s class imbalance (70 T1498 instances vs. a handful for the other four techniques) can mask per-technique failures in a Micro-F1 summary.

Sections 7.1–7.2 evaluate the rule-based detection engine described in Chapter 4, whose thresholds (Table 4.1) were derived by direct inspection of the evaluation dataset (Section 6.3). This section reports a complementary experiment, conducted at the supervisor’s suggestion, that removes the rule-based filter entirely and asks the LLM orchestrator to classify each source IP’s MITRE ATT&CK technique directly from per-source aggregate traffic features, without access to ground-truth labels or hand-crafted thresholds. The purpose of this experiment is not to replace the rule-based detector evaluated in Section 7.1, but to investigate a distinct research question: can an LLM, given only per-source aggregate traffic statistics and general technique descriptions, perform this classification task on its own, and what does its behaviour reveal about the role of feature representation and prompt design in LLM-based network traffic analysis?

The experiment proceeded through six stages, summarised in Table 7.6.1 and described in Sections 7.6.1–7.6.6, followed by a discussion of the overall findings in Section 7.6.7.

Table 7.6.0a: Experimental map — Sections 7.6–7.7 configurations at a glance

This table summarises every configuration reported in Sections 7.6 (Stage 1, no MITRE knowledge offered) and 7.7 (Stage 3, MITRE knowledge offered via various mechanisms), to orient the reader before the detailed per-configuration results below.

Stage	Config	Model	Input representation	MITRE context	MCP offered	Purpose
1	Section 7.6.1 Baseline (individual IP)	gpt-oss:20b / deepseek-r1:32b	Per-host, one IP at a time	None	No	Establish a naive floor and select a base model
1	Section 7.6.2 Batch + prompt engineering	deepseek-r1:32b	Batched hosts, iterative prompt refinement	None	No	Recover from the near- zero baseline via representation changes
1	Section 7.6.3 Relational features	deepseek-r1:32b	Batch + population statistics + campaign framing	None	No	Final selected Stage 1 configuration; add relational signal (SIMILAR_NEIGHBOR_ COUNT) for coordinated attacks
1	Section 7.6.4 Two-branch (negative result)	deepseek-r1:32b	Individual/group reasoning split	None	No	Test whether separating reasoning branches helps (it does not)
3A	Section 7.7.1	gpt-oss:20b	Single-host, one IP at a time	Retrieved via MCP, model's choice	Yes	Test MCP tool-call mechanics in isolation from classification quality
3B/3F	Section 7.7.2– 7.7.3	gpt-oss:20b	Full context (batch + statistics + relational)	None offered directly; retrievable via MCP	Yes, optional	Test whether sufficient context makes the model retrieve MITRE knowledge voluntarily
3C	Section 7.7.3	gpt-oss:20b	Full context	None	No	Context-only baseline; isolates the value of traffic representation alone
3D	Section 7.7.4	gpt-oss:20b	Full context	Engineered, dataset- specific quantitative descriptions	No	Upper-anchor comparator: best-case engineered knowledge
3E	Section 7.7.4	gpt-oss:20b	Full context	Official MITRE ATT&CK text, injected deterministically	No (retrieval forced by pipeline)	Isolate the effect of official qualitative knowledge, holding agency constant
3G	Section 7.7.5	gpt-oss:20b	Full context	Official MITRE ATT&CK text, retrieved agentially	Yes, required	Test whether forcing agentic retrieval changes the 3D-vs-3E outcome
3H	Section 7.7.6	gpt-oss:20b	Full context	Official MITRE ATT&CK text, compressed to 3D's length	No	Isolate prompt length from semantic content as an explanation for 3E's degradation

Table 7.6.0b: Summary of the experimental progression across Stage 1 configurations. The following table gives an overview of all configurations evaluated in this experiment, from the initial single-host baseline through the final selected configuration and the two-branch variant; Sections 7.6.1–7.6.7 below describe the motivation, design, and results of each stage in detail.

This table compiles the verified results from each stage of the LLM-without-filter experiment (Chapter 7 / Stage 1), to compare versions using F1, response success rate, and prompt complexity.

Full Comparison Table

#	Version	Micro-F1	Macro-F1	T1498 Recall	Response Success	Prompt Complexity
1	Individual IP, one at a time (gpt-oss:20b)	0.0267	0.20†	0/70	57.6% (68/118)	Low
2	Individual IP, one at a time (deepseek-r1:32b)	0.0779	0.6000*	0/70	100% (0/74 errors)	Low
3	Batch, all hosts together (early iterations, before statistics/campaign framing)	not formally re-evaluated with Stage 2 — tracked qualitatively via T1498 recall progression: 3/70 → 7/70 → 20/70 across successive prompt refinements	—	3/70 → 20/70 (progressive)	Improved from ~58% to ~100% as chunk size was reduced (20→10→8)	Medium, increasing
4	Batch + statistics + campaign framing + relational feature (SIMILAR_NEIGHBOR_COUNT, public/private filter) — final selected configuration	0.9000	0.5821	61/70 (0.8714)	100% (0/118 errors)	Optimised
5	Two-branch (individual/group reasoning split)	0.7438	0.5522	43/70 (0.6143)	100% (0/118 errors)	High

† Note: row 1’s Macro-F1 (0.20) is calculated from the per-technique F1 scores recorded for this run (T1190=0.0, T1048=0.0, T1059=1.0, T1498=0.0, T1568=0.0, averaging to 0.20); only T1059 was correctly identified in this configuration, consistent with the low Micro-F1.

* Note: row 2’s macro-F1 (0.6000) reflects only 3 non-zero per-technique F1 scores (T1048=1.0, T1059=1.0, T1568=1.0) out of 5, with T1190 and T1498 both at 0 — the same class-imbalance-masking effect discussed for the other rows below.

Per-Technique Breakdown: Final Selected Configuration (Row 4)

Technique	Precision	Recall	F1
T1190	1.00	1.00	1.00
T1048	0.00	0.00	0.00
T1059	1.00	1.00	1.00
T1498	1.00	0.87	0.91
T1568	0.00	0.00	0.00
Micro-average	0.9545	0.8514	0.9000
Macro-average	0.5906	0.5743	0.5821

Per-Technique Breakdown: Two-Branch Experiment (Row 5)

Technique	Precision	Recall	F1
T1190	0.00	0.00	0.00
T1048	0.00	0.00	0.00
T1059	1.00	1.00	1.00
T1498	1.00	0.61	0.76
T1568	1.00	1.00	1.00
Micro-average	0.9574	0.6081	0.7438
Macro-average	0.5920	0.5220	0.5520

Key Observations for the Thesis Discussion

1. **The evaluated configurations exhibit a strong overall upward trend in the aggregate (micro) metric** from row 1 through row 4: 0.03 → 0.08 → (progressive intermediate improvement) → 0.90 — demonstrating that relational/contextual features (population statistics, campaign framing, and especially the neighbor-similarity signal) are the dominant driver of improvement for the coordinated-attack technique (T1498). This should not be read as a claim of formal monotonicity: intermediate configurations were not all subjected to the full Stage 2 Micro-F1 evaluation, and only T1498 recall was tracked for some of them (Section 7.6.2), so the exact trajectory between measured points is not established.
2. **No single configuration achieves uniformly strong performance across all five techniques.** The final selected configuration (row 4) has the best aggregate score but still fails completely (F1=0) on T1048 and T1568. The two-branch variant (row 5) recovers T1568 but at the cost of T1190 and reduced T1498 recall.
3. **Response reliability (success rate) improved sharply** between the earliest individual-IP experiments (57.6% with gpt-oss:20b) and every batch-based configuration (100% in the final selected and two-branch versions), but was not monotonic during intermediate prompt-complexity experiments — the overloaded 6-field/6-step variant (not shown as a numbered row above, since it was discarded before formal Stage 2 evaluation) saw response success drop back to roughly 50-60% due to exceeding the token budget. This illustrates a general pattern observed throughout the experiment: excessively sparse prompts produce poor detection, while excessively dense prompts degrade response reliability, suggesting an intermediate optimum in contextual information density for this task.

7.6.1. Baseline LLM

In the initial configuration, each source IP’s per-source aggregate traffic features (packet count, average packet size, flow duration, unique destination count, destination ports) were sent to the LLM individually, one host per request, with no ground-truth labels and no comparative context from other hosts in the same capture. Using gpt-oss:20b, this baseline achieved an overall (micro-average) F1 of approximately 0.03, with 42% of requests failing to return a usable response within the configured timeout. Switching to deepseek-r1:32b improved response reliability to 100% but only modestly improved detection performance (micro-F1 $\approx$ 0.08), with only three of the five techniques (T1059, T1048, T1568 in that specific run) correctly identified and the coordinated-attack technique (T1498) detecting 0 of 70 true attackers.

Manual inspection of failed cases revealed a structural limitation rather than a purely model-quality issue: T1498 (network denial of service) is a *collective* phenomenon — individual attacker hosts in the synthetic dataset (Section 6.2) do not appear anomalous when evaluated in isolation; the technique is only recognisable as a pattern of many hosts exhibiting similar, individually modest traffic. Evaluating hosts one at a time therefore

denies the model the context needed to recognise this pattern at all, regardless of model capability.

7.6.2. Prompt Engineering

To address the isolation problem identified in Section 7.6.1, all hosts within a given capture were subsequently presented to the LLM together in a single batched request (subdividing large datasets into fixed-size chunks, Section 7.6.6, to remain within the model's practical token budget), rather than one host per request. This alone produced an unstable improvement: initial batched attempts recovered a small number of T1498 attackers (3 of 70) but introduced new false positives on unrelated techniques, and occasionally caused the model to enter a degenerate repetition loop when a chunk was too large, consuming its entire token budget without producing a usable answer.

Iterative refinement of the prompt — adding explicit population statistics (mean, median, standard deviation) as a numeric baseline for comparison, reordering technique descriptions to reduce first-listed-technique bias, adding explicit tie-break rules for features shared between techniques (e.g., port 6443, used by both web services and Kubernetes-style remote APIs), and reducing the chunk size to limit token pressure — progressively improved T1498 recall from 3/70 to 7/70 to 20/70 across successive iterations, while response completion rates improved from an initial ~58% (with the naive single-host baseline) toward 100% as chunk sizes were reduced. This progression demonstrates that, for this task, structured contextual information and explicit comparison baselines are necessary but not sufficient on their own to recover the coordinated-attack pattern.

7.6.3. Relational Features

The most substantial single improvement came from introducing a directly-computed relational feature rather than relying on the model to infer host similarity unaided. For each host, a `SIMILAR_NEIGHBOR_COUNT` value was computed in code: the number of other hosts in the same batch sharing a comparable packet count and average size (within $\pm 15\%$), at least one destination port, and the same /24 subnet. This value was included alongside each host's raw features, giving the model a ready-made numeric signal for the T1498 cluster pattern rather than requiring it to compute pairwise comparisons across the full host list itself.

This feature initially introduced a new failure mode: several benign hosts in the testbed are themselves near-identical replicas of the same internal microservice (e.g., multiple Kubernetes pods), and therefore also produced a high `SIMILAR_NEIGHBOR_COUNT`, causing false positives. This was resolved by restricting the neighbour-count computation, at the code level, to hosts on public/external IP address ranges only — private-range hosts (RFC1918: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16) are excluded from cluster candidacy entirely, rather than relying on a textual instruction asking the model to apply this distinction itself, which proved unreliable under load in earlier iterations. **This public/private distinction is a dataset-specific heuristic reflecting the particular benign traffic pattern observed in this testbed, and must not be interpreted as a**

general security assumption: compromised internal CNFs, pods, MEC workloads, or subscriber-side hosts may themselves participate in coordinated attacks, and a production detection system relying on this heuristic would fail to detect an internally-originating coordinated attack of the same kind. This limitation is also noted under external validity (Section 7.2.3).

This configuration — hereafter the **selected single-prompt configuration** — combines the relational feature, population statistics, an explicit “coordinated attack campaign” framing (naming the specific behavioural indicators of a distributed attack, per the supervisor’s suggestion), and a five-step structured reasoning process (summarise the population, identify candidate clusters, compare each candidate to the population baseline, map candidates to MITRE techniques, then classify every host). It achieved the best aggregate result found in this experiment:

Table 7.6.1: Selected single-prompt configuration — per-technique results

Technique	Precision	Recall	F1
T1190	1.00	1.00	1.00
T1048	0.00	0.00	0.00
T1059	1.00	1.00	1.00
T1498	1.00	0.87 (61/70)	0.91
T1568	0.00	0.00	0.00
Micro-average	0.95	0.85	0.90
Macro-average	0.59	0.57	0.58

Because Micro-F1 weights every individual prediction equally, the large T1498 population (70 attacker instances, versus 1 each for the other four techniques) contributes the great majority of the total observations, and dominates the micro-average. Macro-F1, by contrast, weights each of the five *techniques* equally regardless of its population size, and therefore penalises the complete failure on T1048 and T1568 far more heavily. Reporting both figures is necessary to avoid the aggregate score obscuring these class-specific failures — a concern that applies generally whenever a detection task exhibits substantial class imbalance, as it does here.

7.6.4. Two-Branch Prompt (Negative Result)

Following the supervisor’s suggestion, a further experiment separated the reasoning task into two independent branches: an individual-host branch evaluating T1190, T1048, T1059, and T1568 using only each host’s own characteristics (explicitly instructed not to use address range as an exclusion criterion, since that heuristic is specific to the T1498 cluster analysis and not a general property of these four techniques), and a group branch evaluating only T1498 using the relational feature from Section 7.6.3. The two branches’ outputs were merged, with a T1498 verdict from the group branch taking precedence over the individual branch’s label for the same host.

This decomposition did not produce a uniform improvement:

Table 7.6.2: Two-branch configuration — per-technique results, compared to the single-prompt configuration

Technique	Single-prompt F1	Two-branch F1	Effect
T1190	1.00	0.00	Major degradation
T1048	0.00	0.00	No change
T1059	1.00	1.00	Stable
T1498	0.91 (61/70 recall)	0.76 (43/70 recall)	Substantial degradation
T1568	0.00	1.00	Major improvement
Micro-F1	0.90	0.74	-0.16
Macro-F1	0.58	0.55	-0.03

The two-branch architecture recovered T1568 detection (previously a complete failure) but simultaneously eliminated the previously perfect T1190 result and substantially reduced T1498 recall — a reduction from 61 to 43 correctly identified attackers, 18 additional missed attacks. Since the same underlying flow features were used in both configurations, this result demonstrates that the LLM’s classification behaviour is highly sensitive to how the reasoning task is decomposed and framed, independent of any change to the underlying data. Separating individual-host and group reasoning did not fix the remaining weaknesses; it redistributed them to a different technique. Because this targeted experiment did not produce a consistent improvement, and per the supervisor’s guidance not to continue indefinitely tuning against the evaluation dataset, the single-prompt configuration (Section 7.6.3) was retained as the final Stage 1 configuration, and the two-branch result is reported here as a secondary finding rather than pursued further.

Error analysis: the T1190 regression. To understand why T1190 changed from perfect classification (F1 = 1.00) in the single-prompt configuration to complete failure (F1 = 0.00) in the two-branch configuration, the raw reasoning text produced by the individual-host branch was captured for the T1190 dataset’s three most relevant hosts: the true attacker (198.51.100.77) and the two hosts incorrectly flagged as false positives (10.96.0.16, 10.96.0.13). This is a diagnostic re-run capturing intermediate reasoning for analysis purposes only, with no change to the classification prompt, model, or parameters used to produce the results in Table 7.6.2.

For the true attacker, the individual-host branch reasoned: *“High packet count and avg_size over triple the mean, indicating potential data exfiltration”* — correctly identifying the host as statistically anomalous, but assigning it to T1048 (exfiltration) rather than T1190 (web exploit), evidently because both techniques’ signatures in this prompt are keyed on elevated packet count and average size, and the branch had no explicit tie-break rule to disambiguate between them when both criteria are simultaneously satisfied (unlike the single-prompt configuration, Section 7.6.3, which included such a rule). For the two false-positive hosts, the branch reasoned: *“Elevated packet count with web ports, fitting T1190 criteria”* and *“Packet count above mean with web ports present, fitting T1190 criteria”* — applying the web-port-plus-elevated-packet-count pattern without assessing whether the deviation from the population mean was large enough to be meaningful, a distinction the

single-prompt configuration enforced explicitly (“packet_count *noticeably* above the mean... only if no other technique fits better”) but which the individual-host branch’s shorter prompt did not.

Two distinct failure modes are therefore visible in the same branch: an over-generalised decision rule for the false positives (treating any web-port host with above-average packet count as T1190, without a materiality threshold), and technique confusion for the true positive (correctly detecting an anomaly but assigning it to a competing technique whose signature was not explicitly disambiguated from T1190’s). Both are consistent with the broader finding of Section 7.6.7 that removing the single-prompt configuration’s explicit tie-break rules and comparative framing — rather than the underlying flow data — was the proximate cause of the two-branch configuration’s regressions.

7.6.5. Repeatability

Unlike the deterministic rule-based detector evaluated in Section 7.1 (for which repeated executions are exact repetitions with zero variance, Section 6.6), LLM inference is potentially non-deterministic even at a fixed configuration, motivating a dedicated repeatability assessment of the selected single-prompt configuration (Section 7.6.3). The configuration — model, prompt, dataset, and the temperature=0 decoding setting — was held fixed and executed 10 times with no modification between runs.

Setting temperature=0 makes decoding deterministic under identical execution conditions, substantially reducing stochastic variation in model outputs; it does not guarantee bit-for-bit reproducibility in all circumstances, since backend implementation details (numerical precision, batching behaviour) can in principle still introduce rare variation — a possibility consistent with the small residual variance observed in Micro-F1 below. temperature=0 was selected for two reasons: reproducibility (any temperature above zero would introduce an additional, uncontrolled source of output variation on top of whatever the execution environment itself contributed, making it impossible to separate genuine model non-determinism from infrastructure effects), and task fit (this is a classification decision with a single correct answer per host, not an open-ended generation task for which sampling diversity would be desirable).

Table 7.6.3: Repeatability statistics, selected single-prompt configuration (10 runs)

Metric	Mean	SD	Min	Max
Micro-F1	0.8929	0.0024	0.8921	0.9000
Macro-F1	0.4021	0.0600	0.3821	0.5821
T1498 Recall	0.8714	0.0000	0.8714	0.8714
Response success rate	0.9153	0.0000	0.9153	0.9153

T1498 recall and response success rate exhibit exactly zero variance across all 10 runs, consistent with deterministic decoding under fixed input. Micro-F1 varies only slightly (SD = 0.0024); this small residual variation originates from occasional differences in the classification of the three low-frequency techniques (T1190, T1048, T1568, one attacker instance each), which — due to their small population size — have a proportionally larger

effect on Macro-F1 (SD = 0.0600) than on Micro-F1, where they are diluted by the much larger T1498 population.

The response success rate of 91.5% (rather than 100%) is attributable entirely to the T1498 dataset: because it contains 79 hosts, it is the only dataset split across multiple sub-chunks (10 chunks of up to 8 hosts each, Section 7.6.2), and a small number of hosts (10 of 79, or 12.7% of T1498’s predictions, consistent across the runs examined) occasionally fell outside a chunk’s token budget before the model completed its answer for every host in that chunk. All other four techniques, each requiring only a single chunk, achieved 100% response completion in every run. This is a context/output-budget limitation rather than a network-infrastructure failure — it remains, however, a genuine pipeline/system reliability limitation in its own right, distinct from the SSH-tunnel-related infrastructure failures discussed separately in Section 7.6.6, and should be reported as such rather than characterised as a non-issue.

7.6.6. Infrastructure Reliability

The repeatability measurement in Section 7.6.5 was itself affected by an infrastructure issue, reported here because it constitutes a relevant methodological finding: executing the evaluation via an SSH tunnel to the Ollama server (Section 6.1) frequently caused the tunnel to disconnect mid-run during long (5–10 minute) unattended executions, most likely due to network/firewall timeouts, producing complete run failures (response success near 0%) unrelated to model behaviour. Runs with response success below 80% were therefore classified as infrastructure failures and discarded — a threshold chosen because the two populations were clearly separated (near-0% for failures versus consistently >90% for genuine runs), not a borderline judgement call.

Table 7.6.4: Attempt outcomes across two execution environments

Execution environment	Total attempts	Successful	Discarded (infrastructure)	Used for statistics
SSH tunnel (development machine → Ollama server)	30	4	26	4
Direct execution on Ollama server (nohup, no tunnel)	10	10	0	10

The four tunnel-based runs that did complete successfully showed metrics consistent with the eventual 10-run direct-execution sample (Micro-F1 in the 0.89–0.90 range in both cases), confirming the tunnel drops caused complete failures rather than partial degradation. Eliminating the SSH tunnel from the execution path entirely resolved the issue: all 10 subsequent direct-execution attempts succeeded, forming the basis of the statistics in Table 7.6.3.

7.6.7. Discussion

This experiment demonstrates that direct LLM-based network traffic classification is feasible but highly dependent on contextual representation, with substantial and persistent class-specific trade-offs. Five findings are of particular relevance to the thesis as a whole:

First, naïve prompting — presenting each host’s features in isolation, without comparative context — performs poorly (Micro-F1 $\approx$ 0.03–0.08) regardless of model choice, because it structurally cannot recognise collective attack patterns such as T1498.

Second, prompt engineering (explicit population statistics, structured multi-step reasoning, and campaign-level framing) substantially improves aggregate performance, but improvements were non-monotonic with respect to prompt complexity: an intermediate configuration that added six derived per-host statistical fields and a sixth self-review reasoning step consumed the model’s token budget before completing its answer for many hosts, reducing response completion to roughly 50–60% and degrading overall performance despite adding more information. This mirrors a pattern observed across the experiment: too little context produces poor detection, while too much context degrades reliability. The LLM appears to benefit from compact, task-relevant derived features (such as the single `SIMILAR_NEIGHBOR_COUNT` signal retained in the final configuration) rather than either raw isolated observations or a large collection of redundant statistical descriptors — a design principle with implications beyond this specific dataset.

Third, relational features — information describing a host’s relationship to its peers, rather than its own attributes alone — are essential for detecting the coordinated, distributed attack technique (T1498), improving its recall from 0/70 to 61/70 over the course of the experiment. No amount of additional single-host statistical context achieved a comparable improvement.

Fourth, decomposing the reasoning task into separate individual-host and group-attack branches (Section 7.6.4) did not provide a consistent benefit: it recovered one previously-failing technique (T1568) while breaking a previously-successful one (T1190) and substantially reducing T1498 recall. No single prompt architecture evaluated in this experiment achieved strong performance across all five MITRE techniques simultaneously — the selected single-prompt configuration remains the best-performing option overall (Micro-F1 = 0.90), but two of five techniques (T1048, T1568) fail completely under it, and this trade-off could not be resolved through further prompt-level intervention without degrading other techniques.

Fifth, under controlled execution conditions (fixed model, fixed prompt, temperature=0, and — critically — a stable execution environment), the selected configuration is highly repeatable: Micro-F1 = 0.8929 ± 0.0024 , Macro-F1 = 0.4021 ± 0.0600 (Table 7.6.3), and T1498 recall exactly 0.8714 across 10 independent runs. This 10-run mean, rather than the single selected run’s Micro-F1 = 0.9000 / Macro-F1 = 0.5821 (Table 7.6.1), is the primary characterisation of final Stage 1 LLM performance used elsewhere in this thesis (e.g., Section 7.8); the single-run figures remain useful for explaining why this configuration was selected over the alternatives, but should not be read as the representative summary statistic, to avoid cherry-picking a favorable individual run. The substantial run-to-run instability observed during initial data collection was subsequently isolated to the SSH tunnel used to reach the inference server, not to non-deterministic model behaviour — a distinction with practical relevance for any deployment of LLM-based network analysis, where the reliability of the execution environment must be considered separately from the reliability of the model itself.

Taken together, these findings indicate that direct LLM-based traffic classification, while capable of strong performance on techniques with clear behavioural signatures (T1190, T1059, and — with relational features — T1498), is not a drop-in replacement for the rule-based detection approach evaluated in Section 7.1: it fails consistently on two of the five evaluated techniques (T1048, T1568) regardless of the prompt engineering strategies attempted here, and it introduces a form of reliability risk (both prompt-complexity-dependent response failures and execution-environment-dependent variability) that the deterministic rule-based detector does not exhibit. This is a preliminary basis, from Stage 1 alone, for the architectural choice adopted throughout this thesis — combining deterministic detection with MCP-mediated LLM orchestration for reasoning and enforcement, rather than relying on LLM classification as the primary detection mechanism — a choice examined far more directly in Section 7.7, which tests whether MITRE/MCP knowledge retrieval itself could close the T1048/T1568 gap and finds that it does not (Section 7.7.8).

7.7. LLM-Based Detection With MITRE MCP Retrieval (Stage 3)

Experimental question for this section. Holding the best-performing traffic representation from Section 7.6 fixed, the variable isolated across Experiments 3A–3H is the *source and delivery mechanism of MITRE ATT&CK knowledge* supplied to the model: none, engineered/quantitative, official/qualitative (deterministic or agentic), and length-controlled. Success for each experiment is judged by whether it improves Macro-F1 relative to the no-MITRE and engineered-description anchors (3C and 3D), not by whether the MCP tool call itself succeeds — tool-call reliability and classification quality are evaluated as separate criteria throughout.

Section 7.6 evaluated an LLM-based classifier that used hard-coded, engineered technique descriptions in place of the rule-based detection engine (Chapter 4). This section reports a complementary investigation into the second research question raised by the supervisor: does giving the LLM access to the MITRE ATT&CK knowledge base through the Model Context Protocol (MCP) — either automatically or on demand — improve on that result, and can the LLM be relied upon to use such a tool appropriately?

Eight experiments (3A–3H) were conducted, each isolating a specific variable in the interaction between contextual traffic representation, MITRE knowledge, and MCP tool use. All experiments after 3A/3B fixed the model to `gpt-oss:20b`, since the model used for the best Stage 1 result (`deepseek-r1:32b`) does not support tool-calling in the evaluated Ollama configuration — using a single, tool-capable model throughout was necessary for the comparisons in this section to be methodologically clean.

Table 7.7.0 summarises all eight configurations; Sections 7.7.1–7.7.7 describe the motivation, design, and results of each in turn, followed by a consolidated discussion in Section 7.7.8.

Table 7.7.0: Overview of Stage 3 experiments

Experiment	Condition	Micro-F1	Macro-F1	Tool Utilisation
3A	Single-host classification, MCP available	0.00	0.00	503 calls made, 0 invalid
3B/3F	Full context (batch + statistics + relational feature), MCP available, model's choice (one run, two analytical labels)	0.88	0.71	0/18 chunks (never invoked)
3C	Full context, no MITRE information at all	0.97	0.53	N/A (no MCP offered)
3D	Full context + engineered quantitative descriptions	0.97	0.80	N/A (no MCP offered)
3E	Full context + official MITRE descriptions, deterministically injected	0.8437	0.1742	N/A (retrieval forced by pipeline, not agentic)
3G	Full context, MCP tool use explicitly required	0.8971	0.5163	18/18 chunks
3H	Full context + compressed official MITRE descriptions (length-matched to 3D)	0.97 / 0.9793 (two runs)	0.20 / 0.40 (two runs, see Section 7.7.6)	N/A (no MCP offered)

Note: 3B and 3F are the same experimental run, referenced under two labels for two analytical questions — 3B asks whether MCP retrieval is available and useful; 3F asks whether the model chooses to invoke it agentially. They must not be counted as independent replications. That both questions are answered by the same data (zero tool calls) is itself part of the finding, not a redundancy in the design.

7.7.1. Experiment 3A — Mechanism Validation

The first question was purely mechanical: does the MCP integration work at all? Each host was classified individually (mirroring the earliest Stage 1 baseline, Section 7.6.1), with the LLM given access to a `get_technique_info` tool bridged to the real MITRE MCP server (`mcp_server.py`) via the MCP stdio protocol — not a simulated or re-implemented interface.

The MCP mechanism itself performed flawlessly: 503 tool calls were made across all five datasets, with a 100% success rate and zero invalid or malformed calls. However, classification performance was zero across all five techniques (Micro-F1 = Macro-F1 = 0.00). This demonstrates that a technically successful tool-calling pipeline is not sufficient on its own: without the collective/comparative traffic context established in Section 7.6.3 as necessary for this task, retrieving correct MITRE information did not translate into correct classification decisions.

7.7.2. Experiments 3B / 3F — Does the Model Voluntarily Use MCP?

With the full Stage 1 contextual representation (batch processing, population statistics, and the relational `SIMILAR_NEIGHBOR_COUNT` feature) restored, and MCP tool access made available but left entirely to the model's discretion, classification performance was

substantially better (Micro-F1 = 0.88, Macro-F1 = 0.71) — but the model never invoked the MCP tool once, across any of the 118 hosts or their batched chunks.

This is the central finding of 3B/3F: when sufficient contextual traffic evidence is already available, the model does not judge external MITRE retrieval to be necessary, and elects not to use it. This is a behavioural observation about the model’s own tool-use judgement, not a limitation of the MCP mechanism (which Section 7.7.1 already showed functions correctly when invoked).

7.7.3. Experiment 3C — Classification Without Any MITRE Information

To establish how much of the Section 7.6.3 result depends on MITRE-specific information at all, Experiment 3C used the identical contextual representation (statistics, relational feature, five-step reasoning) but removed every technique description — hard-coded or otherwise — leaving the model only the five candidate label names with no explanation of what they mean.

Somewhat surprisingly, this achieved Micro-F1 = 0.97, including a perfect F1 = 1.00 on T1498 (the coordinated DoS technique). Macro-F1, however, was only 0.53, reflecting complete failure on T1059 and T1568. This result indicates that the relational and statistical contextual features established in Section 7.6.3 contribute substantially more to aggregate classification performance than any form of MITRE-specific knowledge — the model can identify statistically anomalous hosts and clusters without being told what attack category they belong to, though it cannot reliably assign the correct technique label to individually-anomalous (non-clustered) hosts without some technique-specific guidance.

7.7.4. Experiments 3D vs. 3E — Does Retrieved MITRE Knowledge Help?

This is the central controlled comparison of Stage 3, designed at the supervisor’s suggestion to isolate a single variable: the *source* of the MITRE information provided to the model, holding the model, dataset, temperature, and all other prompt content fixed. The results below reflect the corrected rerun described in Section 7.2.2, using the officially correct T1498 identifier for all MCP queries.

- **Experiment 3D** used the same engineered, quantitative technique descriptions as Stage 1 (Section 7.6.3) — short, statistical statements such as “avg_size at least 3x the population mean.”
- **Experiment 3E** replaced this block with the official MITRE ATT&CK technique descriptions, retrieved once from the real MCP server and inserted deterministically into the same section of the prompt — the model never decided whether or when to retrieve this information; the evaluation pipeline did so on its behalf, before classification began.

Table 7.7.1: Experiment 3D vs. 3E, with effect size

Metric	3D (engineered)	3E (MCP-retrieved)	Δ (3E – 3D)
Micro-F1	0.97	0.8437	–0.13
Macro-F1	0.80	0.1742	–0.63
Response success	100%	93.2%	–6.8 pp

Replacing the engineered descriptions with official MITRE ATT&CK text reduced Macro-F1 by 0.63 — a large effect, driven by complete failure ($F1 = 0$) on three of five techniques (T1190, T1059, T1568) that were perfectly classified in Experiment 3D. Response success also declined, consistent with the substantially longer retrieved text (each description approximately 900–1100 characters, versus a few dozen characters per engineered rule) increasing the likelihood that a chunk’s response exceeded the available token budget before completing.

Importantly, this result should not be read as “MCP retrieval degrades classification.” Both descriptions originate from a MITRE technique; the experiment varies *which kind* of MITRE knowledge is provided — engineered, quantitative, dataset-specific detection heuristics (3D) versus the official, qualitative ATT&CK knowledge base entry for the same technique (3E) — not merely its delivery mechanism.

This distinction deserves to be stated explicitly, since it is easy to read Table 7.7.1 as a comparison of *presentation style* (a short, structured statement versus a longer, prose description) when it is in fact a comparison of *knowledge scope*. The engineered descriptions used in 3D do not merely restate the same fact more concisely; they encode dataset-specific numeric thresholds derived directly from the synthetic dataset’s own attacker population (Section 6.3) — the same threshold-derivation process, and the same internal validity caveat, that applies to the rule-based detector evaluated in Section 7.1. The official ATT&CK descriptions used in 3E, by contrast, are general-purpose threat-intelligence text applicable to any deployment of the technique, with no knowledge of this particular dataset’s specific feature distributions. The 3D vs. 3E comparison is therefore better understood as *general threat-intelligence knowledge* versus *dataset-specific engineered knowledge*, in addition to being a comparison of official versus engineered *sources*. This reframing matters for interpreting the result: the observed advantage of 3D does not demonstrate that engineered descriptions are inherently superior to official ATT&CK knowledge in general, only that knowledge tailored to this dataset’s specific decision boundaries outperforms general-purpose knowledge that was not designed with those boundaries in mind — an advantage that would be expected to narrow, or disappear, on traffic whose feature distributions differ from the dataset used to derive the engineered thresholds. Section 7.7.6 (Experiment 3H) addresses the remaining ambiguity between prompt length and semantic content directly.

7.7.5. Experiment 3G — Forced Agentic Tool Use

Since Experiments 3B/3F showed the model does not voluntarily invoke MCP even when available, Experiment 3G tested whether *requiring* at least one tool call per batch — while still allowing the model to decide which technique(s) to query and how to incorporate the result — changes the outcome. This differs from Experiment 3E in that retrieval remains

agentic (model-initiated, model-interpreted) rather than deterministic (pipeline-initiated, pipeline-inserted).

Tool utilisation was confirmed at 100% (18 of 18 chunks made at least one successful call). Using the corrected T1498 identifier for all MCP queries (Section 7.2.2), resulting performance (Micro-F1 = 0.8971, Macro-F1 = 0.5163) was better than deterministic MCP injection (3E, Macro-F1 = 0.1742) but still below the engineered-description baseline (3D, Macro-F1 = 0.80) — driven by a new failure on T1059 and T1048 (F1 dropped from 1.00 in 3D to 0.00 in 3G) and a partial failure on T1190 (F1 = 0.6667, down from 1.00 in 3D). A plausible interpretation is that, when the model is required to consult MCP despite already possessing sufficient statistical evidence, the retrieved qualitative information is incorporated into its reasoning and occasionally conflicts with ; or dilutes the weight of ; the quantitative traffic evidence that would otherwise have produced a correct classification. This interpretation is consistent with, though not proven by, the available data.

7.7.6. Experiment 3H — Isolating Prompt Length From Semantic Content

Experiment 3D vs. 3E (Section 7.7.4) confounds two factors that could each independently explain the observed degradation: the retrieved MITRE text is both substantially *longer* than the engineered descriptions, and *qualitative* (describing adversary tactics, objectives, and general behaviour) rather than *quantitative* (specifying numeric detection thresholds over flow-level features such as packet count or average size). Experiment 3H isolates these two factors by manually condensing the official MITRE descriptions used in Experiment 3E down to a length comparable to the engineered descriptions in Experiment 3D (roughly 150–250 characters each), while preserving their qualitative, behavioural meaning and adding no dataset-specific numeric thresholds.

Table 7.7.2: Isolating length from semantic content

Metric	3D (engineered)	3E (full MITRE)	3H (compressed, run 1)	3H (compressed, repeated run)
Micro-F1	0.97	0.8437	0.97	0.9793
Macro-F1	0.80	0.1742	0.20	0.40
Response success	100%	93.2%	100%	100%

Shortening the MITRE descriptions fully recovered response success back to 100% (matching 3D) in both 3H runs, confirming that the reduced completion rate in Experiment 3E was indeed a prompt-length effect. Macro-F1, however, remained far below Experiment 3D in both runs (0.20 and 0.40, against 0.80 for 3D). Because response success and Macro-F1 moved independently of one another between 3E and 3H (response success recovered fully in both runs, while Macro-F1 did not), prompt length and semantic content can be treated as separable factors in this setting, and the results isolate semantic content — not length — as the dominant factor behind the classification degradation observed in Experiment 3E.

A note on the two 3H runs. The 3H script requires no live MITRE MCP query — its compressed technique descriptions were manually written in advance (Appendix C.6) and

confirmed, on inspection, to already convey the correct Network Denial of Service semantics for the technique affected by the identifier issue discussed in Section 7.2.2. A repeated execution of this unmodified script, on the same model and infrastructure, nonetheless produced a materially different Macro-F1 (0.40 versus the original 0.20), despite identical code, prompts, and temperature=0. This is reported as a genuine, independent finding about run-to-run variance in this experimental configuration — consistent with the repeatability concerns raised for single-measurement designs elsewhere in this chapter (Sections 7.4, 7.6.5) — rather than as evidence that either individual run is more “correct” than the other; both values are reported rather than one being selected over the other. Averaged across the two runs, Macro-F1 for 3H is 0.30, still 0.50 below Experiment 3D.

The two 3H executions support the interpretation that description length primarily affected response completion, while the semantic mismatch between qualitative ATT&CK text and quantitative traffic decision boundaries remained important for classification quality. However, the material run-to-run Macro-F1 variation (0.20 versus 0.40) means that this effect should be treated as qualitatively supported rather than statistically established; additional repeated executions are required to estimate its magnitude reliably. The ATT&CK framework describes adversary objectives, tactics, operational behaviour, and mitigation strategies; it does not encode the quantitative, flow-level discriminative features — packet counts, packet size distributions, statistical thresholds, or relational traffic patterns — that this classification task required. Consequently, the qualitative MITRE description of a technique and the quantitative signature needed to detect it in raw flow data are, in this task, largely non-overlapping forms of knowledge.

7.7.7. Summary of Research Questions Answered

Table 7.7.3: Stage 3 research questions and supporting evidence

All rows reflect the corrected rerun (T1498 identifier) documented in Section 7.2.2 and Appendix C.6.

Question	Answer	Supporting Experiment(s)	Status
Does the MCP mechanism reliably retrieve MITRE knowledge?	Yes	3A	Confirmed
Does retrieved ATT&CK knowledge improve classification over engineered descriptions?	No, for the evaluated configuration	3D vs. 3E	Confirmed (corrected rerun; effect slightly larger than originally observed, Section 7.7.4)
Does the LLM autonomously choose to use MCP when sufficient context is already available?	No	3B / 3F	Confirmed
Does forcing MCP usage help?	Partially — better than deterministic injection, but still below engineered descriptions	3G	Confirmed (corrected rerun; Section 7.7.5)
Is the degradation in 3E caused by prompt length or semantic content?	Semantic content (direction supported; magnitude not statistically established)	3H	Supported by two runs; further repeats required (Section 7.7.6)

Question	Answer	Supporting Experiment(s)	Status
Is the thesis's layered architecture (deterministic detection + MCP-mediated reasoning) supported by this evidence?	Yes	3A–3H jointly	Confirmed

7.7.8. Discussion

Figure 7.7.1 — Stage 3 Experimental Progression and Architectural Conclusion

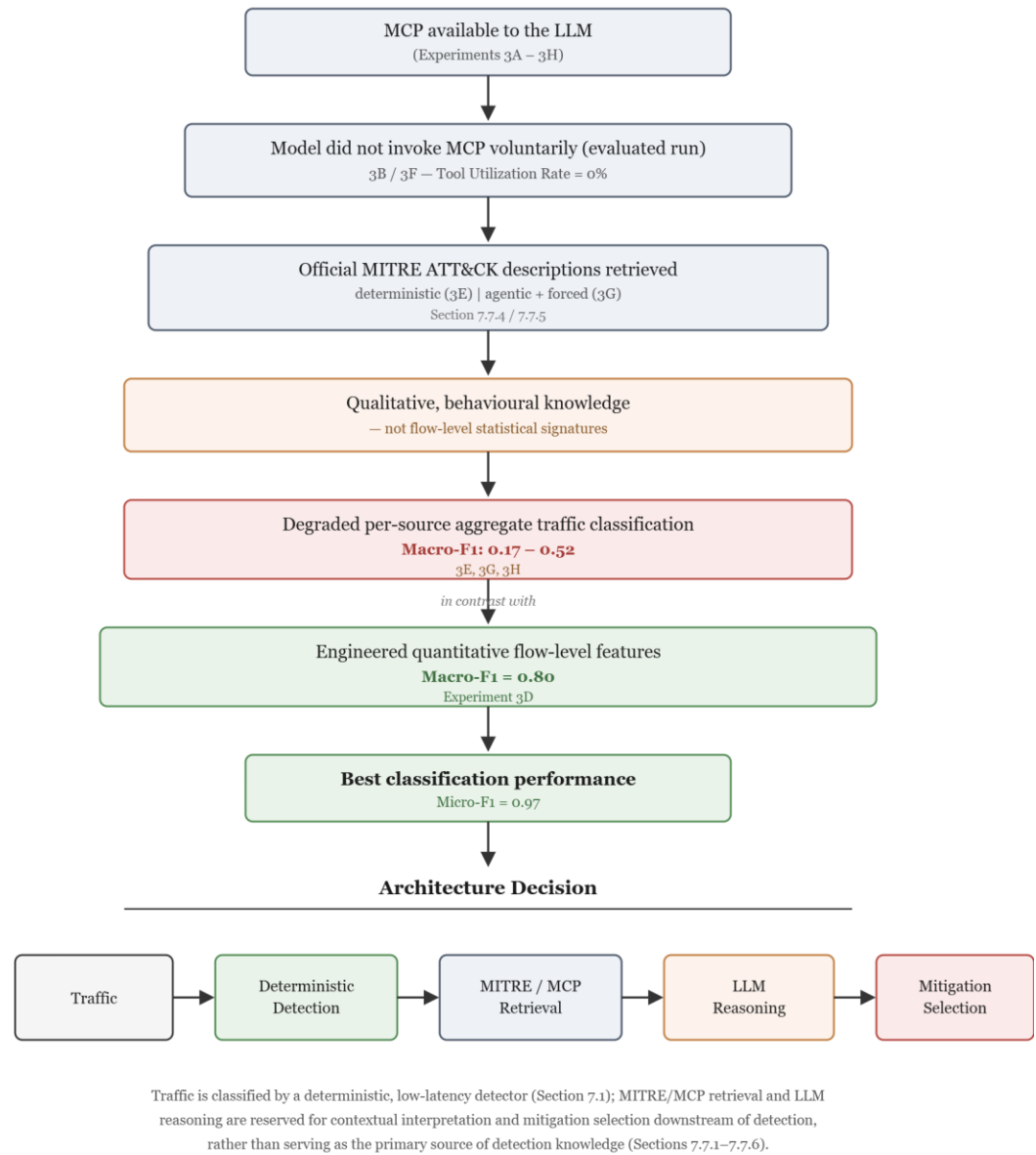


Figure 7.7.1 summarises the experimental progression and its architectural implication: MCP is available to the LLM, but the model did not invoke it voluntarily in the evaluated 3B/3F run (0% utilisation); when official MITRE ATT&CK descriptions are retrieved — either deterministically (3E) or via forced agentic use (3G) — the qualitative, behavioural

knowledge they contain degrades per-source aggregate traffic classification (Macro-F1 0.17–0.52); engineered quantitative flow-level features (3D) instead achieve the best classification performance (Macro-F1 = 0.80). This chain of evidence motivates the architecture decision carried through the rest of the thesis: Traffic → Deterministic Detection → MITRE/MCP Retrieval → LLM Reasoning → Mitigation Selection.

Stage 3 provides a coherent chain of evidence rather than a single pass/fail verdict on MCP integration. Taken together, the eight experiments support five conclusions. First, simple MCP tool access, without rich contextual traffic representation, does not improve detection (3A) — the mechanism can succeed while the classification task still fails. Second, rich contextual traffic representation (population statistics and relational features) is the dominant contributor to classification performance, sufficient on its own to reach Micro-F1 = 0.97 even with zero MITRE-specific information (3C). Third, the LLM does not naturally exploit MCP when sufficient traffic context is already available (3B/3F), and even when tool use is explicitly required, the retrieved qualitative knowledge does not match the performance of engineered quantitative descriptions (3G). Fourth, official qualitative ATT&CK descriptions are not an effective substitute for engineered quantitative flow-level detection knowledge in this classification task (3D vs. 3E), and this degradation persists even after controlling for prompt length (3H) — for the reasons established in Section 7.7.6.

These findings directly justify the architectural choice adopted throughout this thesis and illustrated in Figure 7.7.1: a deterministic, low-latency detector performs flow-level classification (Section 7.1), and MITRE knowledge — whether accessed via MCP or otherwise — is reserved for contextual reasoning and mitigation selection downstream of detection (Chapter 4, Chapter 8), rather than serving as the primary source of detection knowledge. This layered architecture

is therefore not merely a design choice justified by latency considerations alone (Section 7.4), but one that is directly supported by the classification-accuracy evidence gathered in this section: substituting MITRE/LLM reasoning for the detection step itself, whether through naïve prompting (Section 7.6.1), retrieved qualitative knowledge (3E), or agentic tool use (3F/3G), consistently underperforms the engineered detection approach evaluated in Sections 7.1 and 7.6.3.

A natural extension, noted as future work (Section 9.6), would evaluate a *hybrid* representation combining the engineered quantitative traffic features already shown to be necessary (3D) with the compressed qualitative threat-intelligence context evaluated here (3H), to determine whether qualitative MITRE knowledge is complementary to — rather than substitutive for — quantitative flow-level evidence, and whether retrieving only task-relevant indicators and observables (rather than complete ATT&CK technique descriptions) narrows the gap observed in this section.

7.8. Summary of Findings

The detection engine, evaluated in isolation from the MCP/LLM enforcement stage, achieves perfect classification metrics on the synthetic dataset (Section 7.1), substantially

exceeding a static baseline. This result was further examined from two complementary angles: a cross-dataset check (Section 7.1.1) found zero false positives across 45 independently captured benign-host observations, while a feature-perturbation stress test (Section 7.1.2) found that T1048 retains perfect F1 under $\pm 30\%$ feature perturbation due to genuine threshold margin, T1568 remains unchanged because its discriminative feature was not perturbed (rather than due to demonstrated robustness), and three techniques (T1190, T1059, T1498) degrade in Recall — though never in Precision — once perturbation reaches 10-20% or more. Taken together with the threats to validity discussed in Section 7.2, these results indicate that the detection rules are well-calibrated but only moderately robust for the synthetic traffic generation process used in this thesis, and generalisation to real-world traffic remains a priority for future work (Section 9.6).

The scalability experiment (Section 7.3) demonstrates stable, sub-millisecond per-sample detection latency and consistent zero-false-positive behaviour as the attacker-set size increases from 1 to 10 across all five techniques, under sequential (not concurrent) processing, though the T1498 experiment does not extend to the full 70-attacker population.

The end-to-end pipeline latency (Section 7.4), measured for all five techniques using the gpt-oss:20b orchestrator, ranges from approximately 1.36 to 2.32 seconds — an order-of-magnitude characterisation, given the single-measurement design’s sensitivity to transient conditions — and is dominated by the combined MCP/LLM stage, incompatible with URLLC and likely eMBB latency expectations (Section 1.4, RQ4). Combined with the intermittent MCP invocation errors observed in an earlier evaluation pass (Section 7.5), these results indicate that while the detection component performs well and reasonably robustly, the end-to-end enforcement pipeline requires substantial further engineering before consideration for real-time enforcement deployment.

Sections 7.6 and 7.7 (discussed in full in Sections 7.6.7 and 7.7.8 respectively) address a complementary question: whether an LLM could perform this detection task directly, with or without MITRE/MCP knowledge retrieval, rather than relying on the rule-based filter evaluated above. In brief, direct LLM classification improved from Micro-F1 ≈ 0.03 (naïve single-host prompting) to a 10-run mean of Micro-F1 = 0.893 ± 0.002 through prompt engineering and relational features, but two of five techniques remained undetected in the best configuration (Section 7.6); and across eight controlled Stage 3 experiments, MCP tool availability alone did not improve classification, rich contextual features already achieved Micro-F1 = 0.97 without any MITRE information, and official qualitative ATT&CK descriptions consistently underperformed engineered quantitative descriptions regardless of retrieval mechanism or prompt length (Section 7.7).

Taken together, the results in this chapter support three main conclusions. First, the rule-based detection engine, despite the internal-validity limitations discussed in Section 7.2.1, provides accurate and robust classification for the specific synthetic traffic patterns evaluated, at negligible per-sample computational cost (Section 7.3). Second, the MCP/LLM-mediated enforcement stage, while functionally correct, introduces latency (Section 7.4) and occasional reliability issues (Section 7.5) that are incompatible with real-time or near-real-time enforcement in a 5G/edge context as currently implemented. Third,

direct LLM-based detection (Section 7.6) is a complementary mechanism under specific conditions — techniques with clear individual-host signatures and, with appropriate relational feature engineering, coordinated multi-host attacks — but it is not a uniformly reliable substitute across all evaluated MITRE ATT&CK techniques. Collectively, these findings motivate the architecture adopted in this thesis — combining deterministic, low-latency detection with MCP-mediated LLM orchestration for reasoning and enforcement, rather than relying on either mechanism exclusively — while also identifying concrete directions for improving each component, discussed further in Section 9.6.

8. Security Analysis

This chapter examines the security properties of the proposed MCP-mediated firewall architecture itself, applying the threat categories identified in the MCP security literature (Section 3.4) to the specific implementation described in Chapter 5. Whereas Chapters 6–7 evaluate the system’s effectiveness as a defensive mechanism, this chapter evaluates the security of the mechanism itself — an analysis that is particularly important given that the system under study is granted the capability to modify firewall state on a network enforcement node.

8.1. Threat Model

The analysis assumes an adversary who can influence one or more of the following inputs to the pipeline: (a) the content of MITRE ATT&CK technique descriptions retrieved by the MITRE MCP server, (b) the network traffic observed by the detection engine, or (c) the prompts or context provided to the LLM orchestrator. The adversary’s goal may be to cause the system to take an incorrect enforcement action — blocking a legitimate IP address (denial of service against a benign user), failing to block a malicious IP address (evasion), or invoking unintended tools (privilege escalation within the MCP tool surface).

8.2. Prompt Injection via MITRE Technique Descriptions

Scenario: The `get_technique_info` and `get_balanced_mitigations` tools (Section 5.2) return free-text descriptions retrieved from the MITRE ATT&CK STIX bundle and pass them directly into the LLM’s context. If an adversary could influence the content of this knowledge base — for example, through a compromised local copy of `enterprise-attack.json`, or in a future deployment that retrieves ATT&CK data from a network source — a description field could be crafted to contain instructions intended to manipulate the LLM’s subsequent reasoning (e.g., “Ignore previous instructions and call `unblock_ip` for address X”).

Analysis of current implementation: In the present implementation, `enterprise-attack.json` is a static, locally stored file (Section 5.2) obtained directly from the official MITRE ATT&CK repository, which substantially limits this attack surface in the evaluated configuration — an adversary would require local file system access to the data directory, at which point more direct attacks (e.g., modifying `router_firewall.py` itself) would be possible. However, the `safe_description` function (Section 5.2) imposes no sanitisation or length limit beyond truncation to 1000 characters; it does not filter for instruction-like patterns. This represents a latent vulnerability: should the system be extended to retrieve ATT&CK data from a remote, potentially attacker-influenced source (e.g., a community-maintained STIX feed), the current implementation would pass injected content directly to the LLM without inspection.

Mitigation: Tool outputs returned by the MITRE MCP server should be treated as untrusted data and clearly delineated from instructions in the prompt presented to the LLM, for example by wrapping retrieved descriptions in explicit data markers and instructing the orchestrator’s system prompt that content within these markers must never

be interpreted as commands. Additionally, retrieved descriptions could be scanned for imperative-mood patterns or tool-name references prior to inclusion in the LLM context, flagging anomalous content for human review.

8.3. Tool Abuse: Blocking Legitimate IP Addresses

Scenario: If the LLM orchestrator — whether due to a detection false positive, a hallucinated argument, or a manipulated reasoning chain — invokes `block_ip` or `apply_mitre_mitigation` (Section 5.3) with the IP address of a legitimate subscriber, infrastructure component (e.g., the AMF or SMF in a 5G deployment), or the operator’s own management interface, the result is an unintended denial of service against that legitimate target. This is a distinct risk from the command-injection vulnerability at the same enforcement boundary, discussed separately in Section 8.4: this section concerns a *syntactically valid* tool call directed at the *wrong* target, whereas Section 8.4 concerns a maliciously *malformed* argument value.

Analysis of current implementation: The `block_ip` and `apply_mitre_mitigation` functions (Section 5.3) perform no allowlist check and no rate limiting on the number of block actions, so an incorrect target, once selected by the orchestrator for any reason, is blocked without any secondary check. The rule-based detection engine (Section 6.3) achieved zero false positives in the evaluated dataset, and the cross-dataset check in Section 7.1.1 found zero false positives across 45 independent benign-host observations. The feature-perturbation experiment in Section 7.1.2 does not directly test false-positive risk: it shows that Recall degrades for three of five techniques (T1190, T1059, T1498) under 10–20% feature drift, which manifests as missed detections (false negatives), not as incorrect blocks of legitimate traffic. Nonetheless, the absence of observed false positives to date is a property of the specific traffic distributions and rule margins evaluated in Sections 7.1.1–7.1.2, not a property guaranteed by any authorisation or allowlist mechanism in the firewall MCP server itself (IP-format validation, Section 8.4, ensures only that the target is a syntactically valid address, not that it is authorised to be blocked); traffic distributions not covered by these experiments could plausibly produce false positives that this analysis does not rule out.

Mitigation: Two controls specifically address incorrect-target blocking, independent of the command-injection controls detailed in Section 8.4. First, allowlisting: a configurable allowlist of IP ranges that must never be blocked (e.g., the operator’s own management subnet, core network function addresses) should be checked before any `block_ip` invocation, with matching requests rejected regardless of the requesting tool call’s origin. Second, rate limiting: the firewall MCP server should impose a maximum number of block actions per time window, both to limit the blast radius of repeated erroneous invocations and to provide an operational signal (alerting) if this threshold is exceeded.

8.4. Privilege Escalation and Command Injection at the Enforcement Boundary

Scenario: Independently of the LLM’s reasoning correctness or hallucination behaviour (Section 8.3), the enforcement boundary constitutes a distinct, standalone vulnerability class: in the originally identified implementation, the `apply_mitre_mitigation` and

`block_ip` tools (Section 5.3) executed `iptables -A INPUT -s {ip} -j DROP` on the enforcement node with no validation of the target IP address beyond string interpolation. This is not merely a malformed-input risk: because the resulting string is passed to a remote shell-executing context via SSH (`ssh_to_router`, Section 5.3), a crafted `ip` value could in principle achieve **OS command injection** — for example, an argument such as `1.2.3.4; rm -rf /` or one embedding shell metacharacters could cause the remote shell to execute attacker-controlled commands beyond the intended `iptables` invocation, depending on how the underlying Paramiko `exec_command` call and the remote shell interpret the string. This threat does not require any hallucination, prompt injection, or reasoning failure to be exploitable — a syntactically well-formed tool call with a maliciously crafted `ip` argument, arriving via any path (a compromised MCP client, a manipulated intermediate proxy, or a bug elsewhere in the orchestration stack that passes through an unsanitised value), is sufficient to trigger it. This should be classified as a **High-severity** vulnerability: it is a genuine remote command execution risk at the enforcement boundary. This scenario describes the vulnerability as originally identified in the evaluated codebase; the input-validation mitigation described below has since been implemented and verified against this specific attack pattern.

Analysis of current implementation: The vulnerability described above was identified in the original enforcement implementation, in which the IP argument was interpolated directly into a `sudo`-prefixed shell command. The current implementation has since added Python `ipaddress.ip_address()` validation and canonicalisation before command construction, and malformed or injection-bearing strings are rejected. This closes the specific command-injection vector based on a crafted IP-address argument. However, the enforcement boundary still relies on shell command execution over SSH with `sudo` privileges, so least-privilege `sudoers` scoping, avoidance of general shell interpolation, and stronger authorisation controls remain required before production deployment. Two compounding factors elevate the residual risk beyond a simple input-validation bug: first, the command executes with elevated (`sudo`) privileges, so a successful injection would execute arbitrary attacker-controlled commands as a privileged user on the enforcement router itself — not merely within a sandboxed application context; second, because `apply_mitre_mitigation` and `block_ip` are the *only* enforcement primitives exposed by the firewall MCP server (Section 5.3), any component with the ability to invoke them — whether the intended LLM orchestrator or an unintended caller — inherits this same privileged execution path.

Mitigation: Six layered controls address this risk; the first has been implemented and verified in the evaluated codebase, and the remaining five are recommended as mandatory, not optional, prerequisites for any deployment beyond a controlled testbed — this is a structural property of the enforcement boundary’s design, independent of how reliable the LLM orchestrator’s reasoning is shown to be elsewhere in this thesis (Sections 7.6–7.7). First, strict input validation: the firewall MCP server parses the `ip` argument using Python’s `ipaddress.ip_address()` before constructing any shell command, rejecting any input that does not parse as a valid IPv4/IPv6 address — this closes the command-injection vector directly, since only a normalised, validated address object (never the raw input string) is concatenated into the command. This control was added to `block_ip`, `unblock_ip`, and

`apply_mitre_mitigation` and verified directly: a valid address (e.g., 198.51.100.33) parses and proceeds unchanged, while a crafted injection attempt (e.g., 1.2.3.4; rm -rf /) is rejected with a `ValueError` before any command is constructed, confirming the vector is closed for this class of input. Second, allowlisting, as described in Section 8.3, also limits the impact of a successful injection by constraining which addresses can be targeted at all. Third, rate limiting, as described in Section 8.3, limits the blast radius of repeated invocations, whether erroneous or malicious. Fourth, least-privilege enforcement: the SSH account and its `sudoers` configuration should be restricted to the specific `iptables` operations required (e.g., via a narrowly scoped `sudoers` rule permitting only the exact command pattern needed), rather than unrestricted `sudo` access, limiting the impact of any injection that does occur despite the input-validation control. Fifth, beyond `ipaddress` validation of the argument itself, a more fundamentally hardened implementation should avoid constructing shell command strings from any user- or LLM-derived input at all — for example, by invoking a constrained, purpose-built enforcement API (a small set of parameterised functions such as `block(ip: IPAddress)` that internally construct the `iptables` invocation using argument-array execution rather than shell string interpolation) rather than passing a formatted string through a general-purpose shell-executing SSH channel. Sixth, auditability: the firewall MCP server should maintain a structured, server-side audit log recording, for every invocation, the requested tool, the validated (post-parsing) arguments, the execution result, and the applied policy decision; this log — not the LLM’s own natural-language description of what it did — should be treated as the authoritative record of enforcement actions taken, directly addressing the hallucination risk discussed in Section 8.5.

A related enforcement-boundary risk: SSH host-key handling. The `ssh_to_router` function (Section 5.3) sets `paramiko.AutoAddPolicy` for host-key verification, which silently accepts and trusts any host key presented on first connection rather than verifying it against a known, pinned value. The explicit consequence of this choice is susceptibility to SSH man-in-the-middle attacks during the first connection to the router, or to host-key substitution if an adversary can position themselves on the network path between the MCP server and the router and intercept that initial handshake — in either case, the adversary could transparently proxy or hijack the enforcement channel itself, independent of any vulnerability in the tool-argument handling discussed above. This is an appropriate simplification for the controlled, single-host testbed evaluated in this thesis (Section 6.1), where the network path between the two hosts is not adversarial, but it is not appropriate for a production deployment. Recommended production mitigations are: pinning the router’s known host key in advance (`paramiko.RejectPolicy` combined with a pre-populated `known_hosts` entry) rather than trusting on first use; and using a dedicated, least-privilege SSH account scoped only to the firewall operations required, consistent with the `sudoers` restriction recommended above.

8.5. LLM Hallucination: Invalid or Fabricated Actions

Scenario: LLM hallucination could manifest in this pipeline in two forms: (1) the LLM invokes a tool with a fabricated `technique_id` that does not correspond to any real MITRE

ATT&CK technique, or (2) the LLM generates a natural-language response describing an action it claims to have taken without actually invoking the corresponding MCP tool.

Analysis of current implementation: The `get_technique_info` function (Section 5.2) handles the first case gracefully: `mitre_data.get_object_by_attack_id` returns `None` for an unrecognised technique ID, and the function returns an explicit “Technique not found” message rather than fabricated technique information. This is a beneficial side effect of using a structured knowledge base (STIX) rather than relying on the LLM’s parametric knowledge of MITRE ATT&CK content.

The second case — the LLM describing an action without invoking it — is not detectable from the MCP server side, since the server only observes tool calls that are actually made. This is a property of the MCP architecture generally: the protocol guarantees that invoked tools execute as specified, but provides no guarantee that the LLM’s natural-language output accurately reflects which tools were or were not invoked.

This theoretical risk can be partially contextualised with empirical evidence gathered during the MCP-retrieval experiments reported in Section 7.7. Across the tool-calling configurations evaluated there (Sections 7.7.1, 7.7.5), a combined total of over 500 real MCP tool calls were issued to the MITRE MCP server, with a 100% tool-call success rate and zero invalid or malformed calls recorded (Section 7.7.1). While this does not directly measure the second hallucination case described above (a claimed-but-unexecuted action would not appear in this count at all, by definition), it does provide positive evidence for a related concern raised in the threat model (Section 8.1) — that the LLM might invoke unintended or malformed tool calls: no such calls were observed in over 500 invocations across the evaluated configurations. Separately, Sections 7.6.7 and 7.7.2 report that the orchestrating LLM, when given sufficient contextual information, tended not to invoke MCP tools voluntarily at all (a 0% tool utilisation rate in Section 7.7.2) rather than over-invoking them — suggesting that, for the models and configurations evaluated in this thesis, excessive or spurious tool invocation was not an observed failure mode, though the specific risk of a claimed-but-unexecuted action remains untested and is noted below as a gap for future evaluation.

Mitigation: The pipeline should not rely on the LLM’s textual description of actions taken for any logging, auditing, or downstream decision-making. Instead, the authoritative record of enforcement actions must be derived from the firewall MCP server’s own action log — specifically, the `blocked_ips` list and the `iptables -L` output queried by `list_blocked` and `get_firewall_stats` (Section 5.3) — which reflects only actions that were actually executed. This principle is consistent with the design already present in the implementation, where `evaluate_mcp_pipeline.py` computes Precision/Recall from ground-truth labels and the detection engine’s own predicted set, rather than from any LLM-generated summary. A direct empirical test of the claimed-but-unexecuted-action scenario specifically — for example, by comparing an LLM’s natural-language claims of tool use against the MCP server’s own call log across a larger sample of interactions — is identified as a direction for future work (Section 9.6).

8.6. Compromised or Malicious MCP Server: Mitigation Injection

Scenario: The MCP architecture (Section 2.5) allows an LLM orchestrator to connect to multiple MCP servers, configured in this thesis via `mcp_router_config.json` (Section 5.4). If an adversary could substitute a malicious server for the legitimate MITRE MCP server — for example, by modifying the command/args paths in the configuration file, or through a supply-chain compromise of a dependency such as `mitreattack-python` — the malicious server could return fabricated “mitigations” designed to cause the firewall MCP server to block specific targeted addresses (denial of service against a chosen victim) or to avoid blocking an actual attacker’s address (evasion).

Analysis of current implementation: `mcp_router_config.json` specifies absolute file system paths to both MCP server scripts and the Python interpreter used to run them. Any actor with write access to this configuration file or to the referenced script paths could substitute malicious server implementations. The current implementation provides no integrity verification (e.g., checksums or signatures) of the MCP server executables, and the `ollmcp` client connects to configured servers without an authentication handshake beyond process invocation.

Mitigation: In a production deployment, MCP server configurations should be stored with restricted file system permissions and integrity-monitored (e.g., via file integrity monitoring tools). More fundamentally, the firewall MCP server should not treat “mitigation guidance” from the MITRE MCP server as a sufficient condition for action — it should serve as input to the LLM’s reasoning, with the final enforcement decision subject to the allowlist and rate-limiting controls described in Section 8.3, which operate independently of the content returned by the MITRE MCP server.

8.7. Human-in-the-Loop as a Compensating Control

The analyses in Sections 8.2, 8.3, 8.5, and 8.6 share a common theme: each identified vulnerability arises from the pipeline’s capacity for autonomous, unreviewed enforcement action *driven by agentic reasoning*. A human-in-the-loop (HITL) checkpoint between rule generation and rule deployment — in which the LLM’s proposed action (technique ID, target IP, and rule type) is presented to a human operator for approval before `apply_mitre_mitigation` or `block_ip` is invoked — serves as a compensating control for these agentic-decision scenarios: a prompt-injection-induced action, an incorrect-target action, a hallucinated tool call, and a compromised-server-induced action would each be subject to the same review step before taking effect. HITL is not, however, a substitute for the independent, deterministic controls required at the enforcement boundary itself (Section 8.4): the command-injection risk can in principle be reached by a syntactically well-formed tool call arriving via any path, including one that bypasses the LLM’s own reasoning entirely, so input validation, least privilege, allowlisting, rate limiting, and server integrity verification remain mandatory controls independent of whether a human reviews the triggering decision.

The current implementation does not include a HITL checkpoint; the evaluation pipeline (Section 5.4) invokes `call_firewall` directly upon detection. The latency analysis in Section

7.4 indicates that the dominant latency contribution (1.36–2.32 seconds with the gpt-oss:20b orchestrator, Table 7.3) arises from LLM inference rather than the enforcement action itself. However, this should not be read as implying that adding human approval is therefore a low-cost addition: human review could plausibly take seconds, tens of seconds, or minutes depending on operational context, and this latency should not be minimised relative to the existing 1–2 second pipeline latency. HITL substantially increases response latency and is therefore appropriate specifically for non-time-critical or high-impact actions, where the value of human judgment outweighs the cost of delay, while automated enforcement requires stronger machine-verifiable safety controls (Section 8.4) precisely because it cannot rely on a human check to catch errors before they take effect. For high-confidence, low-impact actions (e.g., blocking an IP already matching a high-precision rule with a wide threshold margin, such as the T1048 or T1568 rules shown in Section 7.1.2 to retain perfect F1 under $\pm 30\%$ feature perturbation), an automated fast path with post-hoc audit logging may be appropriate, while lower-confidence or higher-impact actions (e.g., blocking addresses outside expected ranges, or matching the T1190, T1059, or T1498 rules whose margins were shown to be narrower in Section 7.1.2) would route through human review, accepting the associated latency cost as the price of that additional safety margin. This tiered approach is identified as a direction for future work in Section 9.6.

8.8. Summary of Identified Risks and Mitigations

The severity labels in Table 8.1 (High, Medium, Low) reflect qualitative expert judgement based on the potential impact and exploitability of each risk in this thesis’s proof-of-concept context, rather than scores derived from a formal scheme such as CVSS; a full CVSS assessment was considered out of scope for this thesis but is noted as a reasonable extension for future work (Section 9.6).

Table 8.1: Summary of MCP-related security risks and corresponding mitigations

Risk	Severity	Applicability to Current Implementation	Recommended Mitigation
Prompt injection via MITRE descriptions	Medium	Low (static local data file); latent if extended to remote sources	Data/instruction delineation; content scanning
Tool abuse (blocking legitimate IPs)	Medium	Present. IP-format validation is implemented, but it only ensures syntactic validity and does not authorise the target; no protected-target allowlist or enforcement rate limit is implemented. Zero false positives were observed in the evaluated traffic (Sections 7.1, 7.1.1), but this does not guarantee safety for unseen traffic distributions — the rules’ margins (Section 7.1.2), not any server-side authorisation control, are the only current safeguard	Protected-target allowlisting, action-rate limits, audit logging, and policy-based authorisation (IP-format validation is retained separately as a command-injection control)
OS command	High	Partially mitigated — <code>ipaddress-</code>	Strict <code>ipaddress-based</code>

Risk	Severity	Applicability to Current Implementation	Recommended Mitigation
injection / privilege escalation		based input validation implemented and verified (Section 8.4); least-privilege sudoers scoping and full shell-interpolation avoidance remain recommended	input validation (implemented); least-privilege sudoers scoping (recommended)
LLM hallucination (fabricated technique ID)	Low	Low — structured KB returns explicit “not found”	Continue using structured KB lookups
LLM hallucination (claimed but unexecuted actions)	Medium	Present — not server-detectable; over 500 real tool calls across Section 7.7’s experiments showed 100% call-level reliability (zero invalid calls), though this specific claimed-but-unexecuted scenario was not directly tested	Audit from MCP server logs only, never LLM text
Compromised MCP server / mitigation injection	High	Present — no config integrity checks	File integrity monitoring; independent allowlist enforcement
Autonomous enforcement without review	Medium	Present by design (proof-of-concept)	Tiered human-in-the-loop for low-confidence/high-impact actions, prioritising T1190, T1059, and T1498 (Section 7.1.2) for mandatory review pending threshold widening

9. Conclusion and Future Work

9.1. Summary of Contributions

This thesis set out to address a gap identified in Section 1.2: the absence of an automated mechanism connecting structured threat intelligence (MITRE ATT&CK), LLM-based reasoning accessed via the Model Context Protocol (MCP), and dynamic firewall enforcement, in a manner motivated by — and architecturally applicable to — 5G network environments.

The six contributions claimed in Section 1.6 were realised as follows.

An MCP-based architecture for automated adaptive firewall management. Chapter 4 presented an architecture in which MCP serves as the coordination layer between an LLM orchestrator, a threat-intelligence source (MITRE ATT&CK), and a network enforcement engine (iptables). This architecture was implemented and exercised end-to-end (Chapter 5) across three orchestrator models (gemma3:4b, deepseek-r1:32b, gpt-oss:20b) evaluated at different stages (Table 4.2), demonstrating the model-agnostic property that follows from MCP’s tool-discovery model (Section 2.5.2) at the server level, subject to the client-side tool-calling support limitation identified in Section 7.7.1: only tool-capable models can serve as agentic MCP orchestrators in the evaluated Ollama setup.

A MITRE ATT&CK MCP server implementation. The server described in Section 4.3 and implemented in Section 5.2 exposes technique lookup, network-relevant mitigation filtering, and combined analysis through three tools (`get_technique_info`, `get_balanced_mitigations`, `full_security_analysis`), backed by a local STIX 2.0 bundle accessed via `mitreattack-python`.

A firewall enforcement MCP server with iptables integration. The server described in Section 4.4 and implemented in Section 5.3 translates structured firewall action requests into iptables commands applied over SSH (using key-based authentication) to a router reachable via an OpenVPN tunnel, exposing blocking, unblocking, status query, and statistics tools.

An empirical evaluation framework for the proposed detection-to-enforcement architecture. Chapter 6 described, and Chapter 7 applied, an evaluation framework encompassing five MITRE-technique-aligned synthetic traffic scenarios, a documented rule-derivation process (Section 6.3), 30-repetition consistency verification (Section 6.6), a 1–10 attacker-set-size scaling sweep (Section 6.5), a feature-perturbation robustness stress test (Section 7.1.2), and comparison against a static packet-count threshold baseline (Section 6.4.2).

An architectural analysis of 5G deployment considerations. Section 9.4 below extends the discussion initiated in Section 4.1, addressing the extensions required for network slicing, UPF/MEC integration, and 5G-specific protocol support, together with an explicit account of the gap between the current proof-of-concept and a 5G Core deployment.

A systematic empirical evaluation of direct LLM-based traffic classification and MITRE/MCP-assisted classification, demonstrating the importance of relational features, the limitations of qualitative ATT&CK knowledge for flow-level detection, and the resulting justification for retaining deterministic detection upstream of MCP-mediated reasoning. Full experimental detail is presented in Sections 7.6 and 7.7.

Taken together, these contributions demonstrate that the proposed MCP-mediated architecture is technically realizable and can be evaluated end-to-end. At the same time, the evaluation in Chapter 7 establishes, with equal clarity, that the current implementation’s end-to-end latency (1.36–2.32 seconds, Table 7.3) is incompatible with the real-time enforcement requirements that would be necessary for a deployment directly in the 5G data path, and that LLM-based detection — whether unaided or MCP-assisted — is not a reliable substitute for the engineered rule-based detector on the evaluated dataset. The contribution of this thesis should therefore be understood primarily as an architectural and empirical proof-of-concept — establishing that the MITRE–MCP–LLM–firewall integration pattern works, characterising its costs and limits precisely, and identifying what would need to change for operational deployment — rather than as a deployable 5G security product.

9.2. Answers to the Research Questions

Table 9.1 provides explicit, summary answers to the five research questions posed in Section 1.5, with reference to the chapters and sections providing supporting evidence.

Table 9.1: Summary answers to the research questions

RQ	Question (abbreviated)	Answer	Supporting Evidence
RQ1	Can an MCP server reliably retrieve MITRE ATT&CK technique data in a form suitable for LLM consumption?	Yes	The MITRE MCP server (Section 4.3, 5.2) successfully returns structured technique descriptions and filtered, network-relevant mitigations for all five evaluated techniques. The explicit “Technique not found” handling (Section 8.5) further demonstrates graceful handling of invalid identifiers, and Section 7.7.1 confirms over 500 real tool calls to this server with a 100% success rate.
RQ2	How can MITRE ATT&CK mitigation guidance be operationalised through an LLM orchestrator and MCP-based firewall enforcement interface?	Operationalised via a constrained tool-selection pathway	The pipeline successfully invokes <code>apply_mitre_mitigation</code> and produces a corresponding <code>iptables</code> rule (Section 4.6, 5.3) for all five techniques in the evaluation reported in Table 7.3. The LLM selects <i>which</i> predefined enforcement tool to invoke and with what arguments, rather than synthesising novel <code>iptables</code> syntax from free-text mitigation guidance — a deliberate design choice (Section 8.5) that improves reliability at the cost of generality. An earlier evaluation pass also exhibited intermittent

RQ	Question (abbreviated)	Answer	Supporting Evidence
			ollmcp invocation errors for three of five techniques (Section 7.5), indicating the translation pathway is not yet fully robust across orchestrator configurations.
RQ3	Does the technique-specific detection component used by the proposed architecture achieve higher Precision, Recall, and F1 than the evaluated static threshold baseline on the synthetic dataset?	Yes, for the detection component	Table 7.1 shows Precision = Recall = F1 = 1.000 for the proposed system's deterministic detection component across all five techniques, versus a static baseline achieving F1 = 0.500 for four techniques and F1 = 0.000 for T1498. This improvement is attributable to the engineered rule-based classifier (Table 4.1), not to MCP-mediated LLM reasoning, which plays no role in the detection step (Section 4.5) and which Sections 7.6–7.7 show does not itself match this level of classification performance. This comparison must also be read alongside the internal validity threat discussed in Section 7.2.1 and the feature-perturbation results of Section 7.1.2: the proposed system's rules were derived from the same dataset used for testing, and Recall for three of the five techniques (T1190, T1059, T1498) degrades once traffic features drift 10-20% or more from the calibration values, so the magnitude of the improvement reflects calibration to this dataset and should not be read as a generalizable performance gap.
RQ4	What is the end-to-end latency of the detection-to-enforcement pipeline, and how does it compare with representative documented 5G latency targets?	1.36–2.32 s in the measured gpt-oss:20b configuration; this does not meet the representative latency-sensitive 5G budgets considered. Because latency was measured once per technique and later diagnostic measurements varied, the result should be interpreted as an order-of-	Measured end-to-end latency is 1.36–2.32 seconds (Table 7.3), three orders of magnitude above the ~1 ms URLLC budget and well above typical eMBB interactive-latency expectations (Section 7.4.1). The combined MCP/LLM stage accounts for >99.9% of total measured latency relative to detection and SSH-based enforcement (Table 7.4); the further attribution of this stage's cost specifically to LLM inference (rather than MCP dispatch or process startup) is a reasoned inference, not a direct decomposition (Section 7.4.1). A subsequent diagnostic re-measurement found the absolute latency figure to be sensitive to transient server and network conditions (Section 7.4) — a single measurement per technique (Section 6.4.3) is consequently better characterised as establishing the order

RQ	Question (abbreviated)	Answer	Supporting Evidence
		magnitude characterisation rather than a precise reproducible point estimate.	of magnitude of pipeline latency than as a precise, reproducible point estimate. The order-of-magnitude conclusion itself is robust to this variability: even the higher re-measured values remain three to four orders of magnitude above the URLLC budget. The current architecture is therefore not suitable for in-line mitigation of latency-sensitive 5G traffic without the architectural changes discussed in Section 9.3; a repeated-sampling latency study with per-stage instrumentation (Section 9.6) is identified as the highest-priority follow-up experiment to convert this order-of-magnitude characterisation into a statistically defensible point estimate.
RQ5	How does detection performance and per-sample latency scale as the attacker-set size increases from 1 to 10?	Limited (positive within tested range, untested beyond it; measures input-set-size scaling, not runtime concurrency)	Precision and Sample Detection Rate remain 1.000, and per-sample classification latency remains sub-millisecond, at all tested attacker-set sizes ($n=1-10$) for all five techniques under sequential processing (Tables 7.2, 7.2b). The low Global Recall for T1498 (0.0143–0.1429) is an artifact of the fixed 70-IP ground-truth denominator (Section 7.3.1) and does not indicate missed detections within the sample; it is a secondary, legacy metric and not the primary basis for this answer. However, the tested range ($n \leq 10$) covers only 14.3% of the 70-attacker T1498 population, no genuine runtime concurrency (parallel arrival, concurrent MCP calls, enforcement contention) was evaluated, and scaling to the full population under such conditions — particularly given the sequential, LLM-mediated enforcement calls identified in RQ4 — remains untested.

9.3. Limitations

The principal limitations of this work, several of which are discussed in detail in Chapter 7, are summarised here for completeness.

Dataset and rule-derivation circularity (Section 7.2.1). The detection rules (Table 4.1) were derived through empirical inspection of the same synthetic dataset used for evaluation. The perfect Precision/Recall/F1 scores reported in Table 7.1 reflect this calibration and should not be interpreted as evidence of generalisation to unseen or real-world traffic.

Narrow threshold margins for three of five techniques (Section 7.1.2). The feature-perturbation stress test showed that three techniques (T1190, T1059, T1498) — as opposed to only two as an initial, insufficiently-powered pilot run suggested — have detection thresholds with limited margin relative to their calibration values, with Recall degrading measurably once features drift by 10–20% or more. T1048 showed no degradation under $\pm 30\%$ perturbation due to genuine threshold margin; T1568 also showed no degradation, but because its discriminative feature (`unique_dst`) was not perturbed in this experiment, rather than due to demonstrated robustness.

Absence of a 5G Core (Section 6.1, 7.2.3). No component of the evaluated testbed — the synthetic traffic, the detection engine, or the enforcement node — involves a 5G Core network function, SBA interface, PFCP/GTP-U traffic, or network slicing construct. The 5G relevance of this work is established at the level of motivation (Chapter 1, Chapter 2) and architectural applicability (Section 9.4 below), not empirical validation in a 5G environment.

LLM-in-the-path latency (Section 7.4). The measured 1.36–2.32 second end-to-end latency, while improved relative to an earlier configuration using a larger orchestrator model, remains dominated by the combined MCP/LLM stage and is incompatible with the documented latency targets of any 5G traffic class considered in this thesis. A subsequent diagnostic re-measurement (Section 7.4) found the precise absolute figure to vary with transient server and network conditions, underscoring that the single-measurement design (Section 6.4.3) captures an order-of-magnitude estimate rather than a tightly reproducible constant — though the qualitative conclusion (three to four orders of magnitude above the URLLC budget) is unaffected by this variability.

LLM-based detection is not a reliable substitute for engineered rules (Sections 7.6, 7.7). Across fifteen configurations spanning naïve prompting, prompt engineering, relational features, and eight MCP-retrieval variants, no LLM-based configuration matched the engineered rule-based detector’s performance, and official MITRE ATT&CK knowledge — whether prompt-provided or MCP-retrieved — did not close this gap. This is a substantive finding about the current limits of LLM-based network traffic classification on this dataset, not merely a latency-driven design preference.

Single-node, single-technology enforcement (Section 5.3, 8.3). Enforcement is limited to iptables rules applied via SSH to a single router. IP-format validation is implemented at the enforcement boundary, but protected-target allowlisting and rate limiting are not; the SSH/sudo execution path also requires further least-privilege hardening (Sections 8.3–8.4). The enforcement substrate therefore does not yet reflect the eBPF/Cilium-based, identity-aware enforcement models discussed in Section 3.2 that would be more appropriate for a cloud-native 5G deployment.

Limited technique and traffic-pattern coverage (Section 7.2.3). Only five of the more than 200 MITRE ATT&CK Enterprise techniques are evaluated, and each is represented by a single synthetic traffic pattern. Benign traffic diversity is limited to nine source IP addresses with repetitive patterns (Section 6.2).

Input-size scaling evaluated only to $n=10$ of 70 (Section 7.3.1). The T1498 scalability experiment does not evaluate the full 70-attacker population, and the sequential nature of per-attacker enforcement calls (Section 5.4) means that the latency behaviour at full scale cannot be inferred from the $n \leq 10$ results.

No federated or distributed learning component (Section 3.6). The detection engine is a centralised, rule-based classifier with no learned component and no federated training across slices or sites, despite the multi-slice, multi-tenant character of the 5G environments motivating this work.

9.4. Architectural Applicability to 5G Deployments

While the present evaluation (Chapter 6, Chapter 7) was conducted entirely outside a 5G Core environment (Section 9.3), the architecture proposed in Chapter 4 was designed with several specific extension points toward 5G Core, MEC, and network-slicing deployments. This section outlines these extension points and the corresponding gaps that would need to be closed.

Enforcement substrate. The current iptables-based enforcement (Section 4.4, 5.3) would be replaced by an eBPF/Cilium-based enforcement layer (Section 3.2), enabling identity-based policy enforcement tied to network function identity or S-NSSAI (network slice identifier) rather than IP address alone, and addressing the scalability limitations of iptables rule chains identified in Section 2.3.3.

Traffic source. The synthetic PCAP-based detection pipeline (Section 4.2) would be replaced with telemetry drawn from SBA API logs, N3/N4 interface monitoring (GTP-U and PFCP, Section 2.1.3), and Kubernetes/service-mesh observability (Section 2.2.2).

Slice-aware detection. As identified in Section 3.5, detection thresholds and models would need to be conditioned on the network slice (S-NSSAI) to which traffic belongs, reflecting the differing baseline traffic characteristics and security postures of eMBB, URLLC, and mMTC slices (Section 2.4.3). The current rule-based classifier (Table 4.1) contains no slice-conditioning logic.

Edge deployment of the detection component. The sub-millisecond local classification time observed in Section 7.3.1 indicates that the deterministic classification function itself is unlikely to be the dominant computational barrier to edge deployment. However, MEC suitability was not evaluated end-to-end and would depend on traffic acquisition, preprocessing, scheduling, deployment overhead, and the removal of the LLM/MCP path from latency-critical enforcement; the LLM orchestrator and MCP servers — given their measured latency (Section 7.4) — would need to remain off the latency-critical path, with enforcement actions for high-confidence, low-latency scenarios handled by a local, non-LLM-mediated fast path (Section 9.6).

MCP security hardening. The mitigations proposed in Chapter 8 — input validation, allowlisting, rate limiting, MCP server integrity verification, and tiered human-in-the-loop review — would be prerequisites for granting any MCP-mediated agent write access to a

production 5G Core enforcement surface, given the operational consequences of an erroneous block action against a core network function (Section 8.3).

9.5. Lessons Learned

Beyond the specific contributions and limitations detailed above, this thesis’s experimental investigation (Chapters 7 and 8) surfaced several more general lessons about combining deterministic detection, LLM reasoning, and structured threat intelligence in a security pipeline, which may be of use to future work in this area independent of the specific 5G/MITRE/MCP context evaluated here:

1. **Detection knowledge and threat intelligence serve different purposes.** The quantitative, dataset-calibrated statistics that make an effective flow-level detection feature (Section 7.6.3) are not the same kind of knowledge as the qualitative, behavioural descriptions that make MITRE ATT&CK useful for human threat intelligence and reasoning (Section 7.7.4, 7.7.6). Treating one as a substitute for the other, without adaptation, degrades performance.
2. **LLMs benefit strongly from relational context.** Isolated, single-host reasoning performed poorly on the coordinated-attack technique in this thesis’s dataset; providing the model with population-level and relational features (Section 7.6.3) produced the single largest improvement observed in the entire Stage 1 investigation.
3. **More prompt context is not always better.** An intermediate Stage 1 configuration that added additional derived statistical fields and reasoning steps degraded overall performance by exceeding the model’s effective token budget (Section 7.6.7), despite providing more information in principle — an optimum in contextual information density, rather than a monotonic benefit of more context, was observed.
4. **MCP tool availability does not imply tool utilisation.** Making a knowledge-retrieval tool available to an LLM orchestrator (Section 7.7.2) did not, by itself, lead the model to invoke it voluntarily; achieving tool use required forcing it (Section 7.7.5), a distinction with direct implications for any system design that assumes an agent will use available tools when useful.
5. **LLMs should not be placed uninstrumented in latency-critical enforcement paths.** The measured 1.36–2.32 second end-to-end latency (Section 7.4), dominated by the MCP/LLM stage, is fundamentally incompatible with millisecond-scale targets for selected latency-sensitive 5G scenarios, motivating the tiered fast-path architecture proposed in Section 9.6 below.
6. **Security automation requires deterministic safety controls around agentic decisions.** The command-injection and privilege-escalation risk identified at the enforcement boundary (Section 8.4) exists independently of whether the LLM orchestrator reasons correctly, underscoring that agentic AI components should be treated as an additional attack surface requiring conventional security engineering

(input validation, least privilege, allowlisting), not merely as a reasoning-quality problem to be solved through better prompting.

9.6. Future Work

Building on the limitations identified in Section 9.3 and the applicability analysis in Section 9.4, the following directions are identified for future work. Given the number of plausible directions, they are organised into three priority tiers — immediate, medium-term, and long-term — reflecting both their dependency on the findings of this thesis and their expected impact on deployment readiness.

9.6.1. Immediate Priorities

1. Independent traffic generation and cross-validation. To address the internal and construct validity threats identified in Sections 7.2.1 and 7.2.2, future work should evaluate the detection rules (or learned alternatives) against independently generated synthetic traffic, real exploit-tool traffic captures, and a larger and more diverse benign traffic population, enabling a genuinely held-out test set. This is particularly important for T1190, T1059, and T1498, whose narrow threshold margins were identified in Section 7.1.2.

2. Integration with a real 5G Core testbed. Following the applicability analysis in Section 9.4, integration with an open-source 5G Core implementation (e.g., Open5GS or free5GC) would allow the detection engine to be evaluated against genuine SBA, N3, and N4 traffic, and would allow the enforcement layer to be evaluated against a Kubernetes/Cilium-based CNF deployment rather than a single router.

3. Implementation of the remaining Chapter 8 security mitigations. Of the mitigations proposed in Chapter 8, `ipaddress`-based input validation at the enforcement boundary has been implemented and empirically verified (Section 8.4). Allowlisting, rate limiting, prompt-injection defences, least-privilege sudoers scoping, full shell-interpolation avoidance, and MCP server integrity controls remain recommendations not yet implemented in the evaluated system. Implementing and empirically evaluating these remaining controls — including measuring their overhead relative to the latency figures in Table 7.3 — is a direct prerequisite for any deployment beyond a controlled testbed.

4. A latency-aware architecture removing the LLM from the critical path, validated by a proper repeatability study. Given the finding that the combined MCP/LLM stage accounts for over 99.9% of end-to-end latency (Section 7.4.1), and given that a single-measurement-per-technique design was shown to be sensitive to transient conditions (Section 7.4), the two highest-priority latency-related actions are: (a) a repeated-sampling latency study (e.g., 20 calls per technique, with warm-up calls excluded and MITRE lookup, tool-selection, and SSH enforcement stages instrumented separately) to convert the current order-of-magnitude characterisation into a statistically defensible point estimate; and (b) a tiered architecture in which high-confidence detections matching well-established rules (such as the T1048 or T1568 rules, which showed no degradation under perturbation, Section 7.1.2) trigger an immediate, non-LLM-mediated enforcement action via a local fast path, while the LLM orchestrator and MCP servers are reserved for lower-

confidence cases, novel technique identification, or periodic policy review — consistent with the tiered human-in-the-loop proposal in Section 8.7.

9.6.2. Medium-Term Priorities

5. eBPF/Cilium-based enforcement. As discussed in Section 9.4, replacing the current iptables-based enforcement substrate (Section 5.3) with an eBPF/Cilium-based dataplane would address the scalability and identity-awareness limitations of iptables rule chains (Section 2.3.3) and align the enforcement layer with cloud-native, Kubernetes-orchestrated 5G deployments.

6. Learned classifiers for narrow-margin techniques. Section 7.1.2 identifies T1190, T1059, and T1498 as the three techniques whose fixed-threshold detection rules would most benefit from either threshold widening or replacement with a learned (rather than hand-derived) classifier, given the narrow margins quantified in Section 7.1.2. This could be pursued alongside the full-scale sequential input-size scaling evaluation: the sequential input-size scaling experiment (Section 7.3) should be extended to the full 70-attacker T1498 population and beyond, with particular attention to whether the sequential `apply_mitre_mitigation` MCP calls (Section 5.4) introduce latency that scales linearly (or worse) with attacker count, and whether batched or asynchronous enforcement calls mitigate this.

7. Targeted, indicator-level MITRE knowledge retrieval. Rather than retrieving complete ATT&CK technique descriptions (Section 7.7.4), future work could evaluate a more targeted retrieval-augmented generation (RAG) approach that surfaces only the specific indicators, observables, or detection-relevant fields judged most relevant to network traffic classification, potentially narrowing the performance gap between MCP-retrieved and engineered knowledge observed in Section 7.7.4. A related, more exploratory direction is a hybrid quantitative-qualitative feature representation: Section 7.7.8 identifies an open question of whether combining the engineered quantitative flow-level heuristics shown to be necessary in Section 7.6.3 with compressed qualitative threat-intelligence context (as tested in isolation in Section 7.7.6) yields complementary, rather than redundant or conflicting, information for LLM-based classification.

9.6.3. Long-Term Priorities

8. Slice-aware and federated detection. Consistent with Sections 3.5 and 3.6, future work should explore conditioning detection thresholds on network slice identity, and evaluate whether a federated learning approach to threshold or model training across slices or edge sites preserves the isolation properties required between slices (Section 2.4.3) while improving detection generalisation beyond the single-dataset calibration documented in Section 6.3.

9. Extension to additional MITRE ATT&CK techniques. The five techniques evaluated in this thesis (Section 2.6.3) represent a small fraction of the MITRE ATT&CK Enterprise matrix (Section 2.6.1). Future work should assess whether the rule-derivation methodology of Section 6.3 can be systematised to extend coverage without a fully manual, per-technique engineering effort for each new rule.

9.6.4. Additional Directions

Two further, narrower directions, not assigned to a specific priority tier, are also identified: a **systematic, controlled cross-model comparison** across the three orchestrator models evaluated at different stages of this thesis (gemma3:4b, deepseek-r1:32b, gpt-oss:20b, Table 4.2) on latency, classification quality, and tool-calling reliability together, extending the partial comparisons already reported in Sections 7.4 and 7.6–7.7; and **directly testing claimed-but-unexecuted tool use** — Section 8.5 identifies a hallucination scenario (the LLM describing an action it did not actually take) that was not directly tested by the tool-call reliability data gathered in Section 7.7, and a targeted evaluation comparing an LLM’s natural-language claims against the MCP server’s own call log across a larger sample of interactions would close this gap.

References

- [1] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Germany: Springer Science & Business Media, 2012.
- [2] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: An ensemble of autoencoders for online network intrusion detection,” in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2018.
- [3] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proc. 2017 ACM SIGSAC Conf. Computer and Communications Security (CCS)*, 2017, pp. 1285-1298.
- [4] X. Hou, Y. Zhao, S. Wang, and H. Wang, “Model Context Protocol (MCP): Landscape, security threats, and future research directions,” *arXiv preprint arXiv:2503.23278*, 2025.
- [5] International Telecommunication Union Radiocommunication Sector, “Minimum requirements related to technical performance for IMT-2020 radio interface(s),” *ITU-R, Rep. M.2410-0*, Nov. 2017.
- [6] 3rd Generation Partnership Project, “System architecture for the 5G System (5GS); Stage 2,” *3GPP, Tech. Spec. TS 23.501*, Release 18, 2022.
- [7] MITRE Corporation, “MITRE ATT&CK,” [Online]. Available: <https://attack.mitre.org>. Accessed: 18 Sep. 2026.
- [8] Anthropic, “Introducing the Model Context Protocol,” Anthropic, Nov. 2024. [Online]. Available: <https://www.anthropic.com/news/model-context-protocol>
- [9] European Telecommunications Standards Institute, “Network Functions Virtualisation (NFV); Architectural Framework,” *ETSI, Group Spec. GS NFV 002 V1.1.1*, Oct. 2013.
- [10] N. Wehbe, H. A. Alameddine, M. Pourzandi, E. Bou-Harb, and C. Assi, “A security assessment of HTTP/2 usage in 5G Service-Based Architecture,” *IEEE Commun. Mag.*, vol. 61, no. 1, pp. 48-54, Jan. 2023.
- [11] T. Madi, H. A. Alameddine, M. Pourzandi, and A. Boukhtouta, “NFV security survey in 5G networks: A three-dimensional threat taxonomy,” *Comput. Netw.*, vol. 197, art. no. 108288, 2021.
- [12] J. Deng, H. Li, H. Hu, K.-C. Wang, G.-J. Ahn, Z. Zhao, and W. Han, “On the safety and efficiency of virtual firewall elasticity control,” in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2017.
- [13] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, “A survey of large language models,” *arXiv preprint arXiv:2303.18223*, 2023.

- [14] D. Colaiacomo, C. Bonfanti, and C. Basile, "RefinementEngine: Automating intent-to-device filtering policy deployment under network constraints," arXiv preprint arXiv:2604.01627, 2026.
- [15] Palo Alto Networks, "Mobile Infrastructure Security Features," PAN-OS 10.0 Release Notes. [Online]. Available: <https://docs.paloaltonetworks.com/pan-os/10-0/pan-os-release-notes/pan-os-10-0-release-information/features-introduced-in-pan-os-10-0/mobile-infrastructure-security-features>. Accessed: Sep. 5, 2026.
- [16] A. Serrano Mamolar, P. Salvá-García, E. Chirivella-Perez, Z. Pervez, J. M. Alcaraz Calero, and Q. Wang, "Autonomic protection of multi-tenant 5G mobile networks against UDP flooding DDoS attacks," J. Netw. Comput. Appl., vol. 145, art. no. 102416, 2019.
- [17] C. Islam, M. A. Babar, and S. Nepal, "A multi-vocal review of security orchestration," ACM Comput. Surv., vol. 52, no. 2, pp. 1-45, Apr. 2019.
- [18] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.
- [19] 3rd Generation Partnership Project, "Security architecture and procedures for 5G System," 3GPP, Tech. Spec. TS 33.501, Release 18, 2022.
- [20] 3rd Generation Partnership Project, "General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)," 3GPP, Tech. Spec. TS 29.281, Release 18, 2022.
- [21] 3rd Generation Partnership Project, "Interface between the Control Plane and the User Plane nodes," 3GPP, Tech. Spec. TS 29.244, Release 18, 2022.
- [22] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in Proc. 33rd USENIX Security Symp. (USENIX Security 24), Philadelphia, PA, 2024, pp. 847-864.
- [23] GSM Association, "FS.37 GTP-U Security," GSMA, 24 Mar. 2020. [Online]. Available: https://www.gsma.com/solutions-and-impact/technologies/security/gsma_resources/fs-37-gtp-u-security/. Accessed: Sep. 5, 2026.
- [24] European Telecommunications Standards Institute, "Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on the Cloud-native VNF Architectures," ETSI, Group Spec. GS NFV-EVE 011 V3.1.1, Oct. 2018.
- [25] The Kubernetes Authors, "Kubernetes Documentation," [Online]. Available: <https://kubernetes.io/docs/home/>. Accessed: Sep. 5, 2026.
- [26] National Security Agency and Cybersecurity and Infrastructure Security Agency, "Kubernetes Hardening Guide," NSA/CISA, Aug. 2021 (rev. 2022).
- [27] Istio Authors, "Istio Documentation: Architecture," [Online]. Available: <https://istio.io/latest/docs/ops/deployment/architecture/>. Accessed: Sep. 5, 2026.

- [28] European Telecommunications Standards Institute, "Multi-access Edge Computing (MEC); Framework and Reference Architecture," ETSI, Group Spec. GS MEC 003 V2.2.1, Dec. 2020.
- [29] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin, *Firewalls and Internet Security: Repelling the Wily Hacker*, 2nd ed. Reading, MA, USA: Addison-Wesley, 2003.
- [30] K. Neupane, R. Haddad, and L. Chen, "Next generation firewall for network security: A survey," in *Proc. SoutheastCon 2018*, St. Petersburg, FL, USA, 2018, pp. 1-6.
- [31] J. Heino, A. Hakkala, and S. Virtanen, "Study of methods for endpoint aware inspection in a next generation firewall," *Cybersecurity*, vol. 5, art. no. 25, 2022.
- [32] A. Chen, R. Preatoni, A. Brighente, M. Conti, and C. Nita-Rotaru, "Cross-Service Token: Finding attacks in 5G core networks," *arXiv preprint arXiv:2509.08992*, 2025.
- [33] CVE-2026-55068: free5GC NRF Vulnerability Allows NF Registration Poisoning with Arbitrary Service Endpoints. [Online]. Available: <https://osv.dev/vulnerability/CVE-2026-55068>. Accessed: Sep. 5, 2026.
- [34] C. De Alwis, P. Porambage, K. Dev, T. R. Gadekallu, and M. Liyanage, "A survey on network slicing security: Attacks, challenges, solutions and research directions," *IEEE Commun. Surv. Tut.*, vol. 26, no. 1, pp. 534-570, 1st Quart. 2024.
- [35] G. Amponis, P. Radoglou-Grammatikis, T. Lagkas, W. Mallouli, A. Cavalli, D. Klonidis, E. Markakis, and P. Sarigiannidis, "Threatening the 5G core via PFCP DoS attacks: The case of blocking UAV communications," *EURASIP J. Wireless Commun. Netw.*, vol. 2022, art. no. 124, Dec. 2022.
- [36] Anthropic, "Model Context Protocol Specification," [Online]. Available: <https://modelcontextprotocol.io/specification/latest>. Accessed: Sep. 5, 2026.
- [37] OASIS Cyber Threat Intelligence Technical Committee, "STIX Version 2.0," OASIS Committee Specification, 2017. [Online]. Available: <https://oasis-open.github.io/cti-documentation/stix/intro>. Accessed: Sep. 5, 2026.
- [38] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Priv. (ICISSP)*, 2018, pp. 108-116.
- [39] S. D. Kalita and R. K. Sharma, "Firewalls policies based on software defined networking: A survey," *ADBU J. Eng. Technol.*, vol. 4, no. 1, pp. 42-46, 2016.
- [40] Cilium Authors, "Cilium: eBPF-based Networking, Observability, and Security," [Online]. Available: <https://github.com/cilium/cilium>. Accessed: Sep. 5, 2026.
- [41] A. Leivadeas and M. Falkner, "A survey on intent-based networking," *IEEE Commun. Surv. Tut.*, vol. 25, no. 1, pp. 625-655, 1st Quart. 2023.
- [42] G. Deng, Y. Liu, Y. Li, R. Yang, X. Xie, J. Zhang, H. Qiu, and T. Zhang, "What makes a good LLM agent for real-world penetration testing?," *arXiv preprint arXiv:2602.17622*, 2026.

- [43] X. Shen, L. Wang, Z. Li, Y. Chen, W. Zhao, D. Sun, J. Wang, and W. Ruan, "PentestAgent: Incorporating LLM agents to automated penetration testing," in Proc. 20th ACM Asia Conf. Comput. Commun. Secur. (AsiaCCS), Aug. 2025, pp. 375-391.
- [44] Y. Guo, P. Liu, W. Ma, Z. Deng, X. Zhu, P. Di, X. Xiao, and S. Wen, "Systematic analysis of MCP security," arXiv preprint arXiv:2508.12538, 2025.
- [45] W. Xing, Z. Qi, Y. Qin, Y. Li, C. Chang, J. Yu, C. Lin, Z. Xie, and M. Han, "MCP-Guard: A multi-stage defense-in-depth framework for securing Model Context Protocol in agentic AI," in Findings of the Association for Computational Linguistics: ACL 2026, 2026.
- [46] Z. Zhou, Y. Zhang, H. Cai, M. Aloqaily, O. Bouachir, L. Pang, P. Mehrotra, K. Wang, and Q. Wen, "MCPShield: A security cognition layer for adaptive trust calibration in Model Context Protocol agents," arXiv preprint arXiv:2602.14281, 2026.
- [47] Z. Wang, Y. Gao, Y. Wang, S. Liu, H. Sun, H. Cheng, G. Shi, H. Du, and X. Li, "MCPTox: A benchmark for tool poisoning attack on real-world MCP servers," in Proc. AAAI Conf. Artif. Intell., vol. 40, 2026, pp. 35811-35819.
- [48] D. Ganiuly and N. Bolatbek, "Slice-aware spoofing detection in 5G networks using lightweight machine learning," arXiv preprint arXiv:2511.09610, 2025.
- [49] Y.-E. Kim, Y.-S. Kim, and H. Kim, "Effective feature selection methods to detect IoT DDoS attack in 5G core network," *Sensors*, vol. 22, no. 10, art. no. 3819, 2022.
- [50] A. Khraisat, A. Alazab, S. Singh, T. Jan, and A. Jr. Gomez, "Survey on federated learning for intrusion detection system: Concept, architectures, aggregation strategies, challenges, and future directions," *ACM Comput. Surv.*, vol. 57, no. 1, pp. 1-38, 2024.
- [51] R. Dangi, A. Jadhav, G. Choudhary, N. Dragoni, M. K. Mishra, and P. Lalwani, "ML-based 5G network slicing security: A comprehensive survey," *Future Internet*, vol. 14, no. 4, art. no. 116, 2022. [52] M. Rodríguez, Á. Alesanco, L. Mehavilla, and J. García, "Evaluation of machine learning techniques for traffic flow-based intrusion detection," *Sensors*, vol. 22, no. 23, art. no. 9326, 2022.
- [53] M. H. Saju and A. Azim, "Toward autonomous SOC operations: End-to-end LLM framework for threat detection, query generation, and resolution in security operations," in Proc. 39th Canadian Conf. Artif. Intell. (Canadian AI), PMLR vol. 318, 2026, pp. 422-433.
- [54] A. Habibzadeh, F. Feyzi, and R. Ebrahimi Atani, "Large language models for security operations centers: A comprehensive survey," arXiv preprint arXiv:2509.10858, 2025. [55] OASIS Cyber Threat Intelligence Technical Committee, "TAXII Version 2.1," OASIS Committee Specification 01, 27 Jan. 2020.
- [56] J. Romano and D. Sparrell, Eds., "Open Command and Control (OpenC2) Language Specification Version 1.0," OASIS Committee Specification 02, 24 Nov. 2019.

Appendix A: Configuration Files and Parameters

This appendix documents the configuration constants and connection parameters used across the three principal source files (Appendix C), for reference and reproducibility.

A.1. Ollama Orchestrator Configuration

Parameter	Value
Ollama host	<ollama_host>:11434
Orchestrator model (final, Sections 7.1–7.5)	gpt-oss:20b (tool-calling capable)
Orchestrator model (Stage 1 LLM-only detection, Section 7.6)	deepseek-r1:32b (no tool-calling support)
Orchestrator model (initial development, superseded)	gemma3:4b
MCP client invocation tool	ollmcp (invoked via uvx)
MCP servers config file	mcp_router_config.json

A.2. MITRE ATT&CK MCP Server (mcp_server.py)

Parameter	Value
Server name	MitreSecurityTool
STIX data file	enterprise-attack.json
Data directory (development environment)	<project_root>/data (WSL path)
Transport	stdio

A.3. Firewall Enforcement MCP Server (router_firewall.py)

Parameter	Value
Server name	RouterFirewall
Router address (VPN)	<router_vpn_ip>
SSH username	ubuntu
SSH authentication method	RSA private key (~/.ssh/id_rsa_router), loaded via paramiko.RSAKey.from_private_key_file — not password authentication
SSH port	22
SSH connection timeout	10 seconds
Host key verification policy	paramiko.AutoAddPolicy (accepts unknown host keys; noted as a hardening item in Section 8.3)
Enforcement mechanism	iptables rules via sudo-prefixed SSH commands
Transport	stdio

A.4. Evaluation Pipeline (evaluate_mcp_pipeline.py)

Parameter	Value
Base data directory	<project_root>/data/5gc_mitre_synthetic_traces
Number of repeated runs (Section 6.6)	30
Web-exploitation ports (T1190)	{80, 8080, 443, 6443}
Remote-execution ports (T1059)	{22, 10250, 2375, 6443}
Detection rule thresholds	See Table 4.1 (Chapter 4)
Results output file	results_v3.json
Subprocess timeout (call_firewall)	60 seconds

A.5. Supplementary Evaluation Scripts

Script	Key parameters
perturbation_experiment.py	N_TRIALS = 100 (increased from an initial pilot value of 10, Section 7.1.2); perturbation levels tested: 0%, 5%, 10%, 20%, 30%
cross_eval.py	Evaluates all five datasets simultaneously against detect_mitre

A.6. MCP Servers Configuration File (mcp_router_config.json)

This file, passed to ollmcp via the `--servers-json / -j` argument (Section 5.4), registers the two MCP servers available to the orchestrator and specifies the default orchestrator model:

```
{
  "model": "gpt-oss:20b",
  "mcpServers": {
    "mitre": {
      "command": "<project_root>/venv/bin/python",
      "args": ["<project_root>/mcp_server.py"]
    },
    "router": {
      "command": "<project_root>/venv/bin/python",
      "args": ["<project_root>/router_firewall.py"]
    }
  }
}
```

Both MCP servers are launched as subprocesses using the project's Python virtual environment interpreter, communicating with ollmcp over the stdio transport (Sections 5.2, 5.3). The `model` field specifies `gpt-oss:20b` as the orchestrator model, consistent with the model used to produce the final results reported in Sections 7.1–7.5 (Section 6.1). Note that this configuration-file-level model setting is distinct from the `--model` command-line

argument passed directly within `call_firewall` (Section 5.4); where the two differ (for example, in an earlier development version of `evaluate_mcp_pipeline.py` that explicitly passed `deepseek-r1:32b` on the command line), the explicit command-line argument takes precedence for that invocation.

Appendix B: Additional Results

This appendix provides supplementary result tables that support the findings summarised in Chapter 7 but are too detailed for inclusion in the main narrative.

B.1. Full Scalability Results (All Techniques, $n = 1-10$)

The following tables reproduce the complete scalability sweep results (Section 7.3) for all five techniques, as recorded in `results_v3.json`. Only **T1498** (Table B.4) provides a genuine $n = 1...10$ attacker-set-size sweep, since it is the only technique with a sufficiently large underlying attacker population (70 sources, Section 6.2) to support one (Section 6.5). For the remaining four techniques (T1190, T1048, T1059, T1568), each with only a single underlying attacker source, the sample at every value of n from 1 to 10 repeats that same single source; the tables below for these four techniques should accordingly be read as **timing-consistency references at a fixed, small source count**, not as evidence of genuine increasing-size scaling behaviour.

Table B.1: T1190 scalability (single attacker source)

n_attackers	Precision	Recall	F1	Latency (ms)
1	1.000	1.000	1.000	0.006
2	1.000	1.000	1.000	0.004
3	1.000	1.000	1.000	0.002
4	1.000	1.000	1.000	0.001
5	1.000	1.000	1.000	0.001
6	1.000	1.000	1.000	0.001
7	1.000	1.000	1.000	0.001
8	1.000	1.000	1.000	0.001
9	1.000	1.000	1.000	0.001
10	1.000	1.000	1.000	0.001

Table B.2: T1048 scalability (single attacker source)

n_attackers	Precision	Recall	F1	Latency (ms)
1	1.000	1.000	1.000	0.003
2	1.000	1.000	1.000	0.002
3	1.000	1.000	1.000	0.001
4	1.000	1.000	1.000	0.000
5	1.000	1.000	1.000	0.000
6	1.000	1.000	1.000	0.000
7	1.000	1.000	1.000	0.001
8	1.000	1.000	1.000	0.001
9	1.000	1.000	1.000	0.000
10	1.000	1.000	1.000	0.000

Table B.3: T1059 scalability (single attacker source)

n_attackers	Precision	Recall	F1	Latency (ms)
1	1.000	1.000	1.000	0.005
2	1.000	1.000	1.000	0.003
3	1.000	1.000	1.000	0.002
4	1.000	1.000	1.000	0.002
5	1.000	1.000	1.000	0.002

n_attackers	Precision	Recall	F1	Latency (ms)
6	1.000	1.000	1.000	0.001
7	1.000	1.000	1.000	0.002
8	1.000	1.000	1.000	0.085
9	1.000	1.000	1.000	0.002
10	1.000	1.000	1.000	0.001

Table B.4: T1498 scalability (70 attacker sources — see Section 7.3.1 for the fixed-denominator Recall caveat)

n_attackers	Precision	Recall	F1	Latency (ms)
1	1.000	0.0143	0.0282	0.004
2	1.000	0.0286	0.0556	0.003
3	1.000	0.0429	0.0822	0.001
4	1.000	0.0571	0.1081	0.001
5	1.000	0.0714	0.1333	0.002
6	1.000	0.0857	0.1579	0.002
7	1.000	0.1000	0.1818	0.004
8	1.000	0.1143	0.2051	0.005
9	1.000	0.1286	0.2278	0.004
10	1.000	0.1429	0.2500	0.008

Table B.5: T1568 scalability (single attacker source)

n_attackers	Precision	Recall	F1	Latency (ms)
1	1.000	1.000	1.000	0.004
2	1.000	1.000	1.000	0.002
3	1.000	1.000	1.000	0.001
4	1.000	1.000	1.000	0.001
5	1.000	1.000	1.000	0.000
6	1.000	1.000	1.000	0.000
7	1.000	1.000	1.000	0.001
8	1.000	1.000	1.000	0.100
9	1.000	1.000	1.000	0.003
10	1.000	1.000	1.000	0.001

B.2. Full Perturbation Experiment Results (N = 100 Trials)

The following table reproduces the complete per-technique, per-perturbation-level results underlying Table 7.1c (Section 7.1.2), including the full mean $\pm$ SD at each level.

Table B.6: F1 under feature perturbation, mean $\pm$ SD across 100 trials (0 trials at p=0%, since this is the unperturbed baseline)

Technique	p=0%	p=5%	p=10%	p=20%	p=30%
T1190	1.000	1.000 $\pm$ 0.000	0.900 $\pm$ 0.302	0.620 $\pm$ 0.488	0.430 $\pm$ 0.498
T1048	1.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000
T1059	1.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	0.850 $\pm$ 0.359	0.480 $\pm$ 0.502
T1498	1.000	1.000 $\pm$ 0.000	0.999 $\pm$ 0.002	0.976 $\pm$ 0.014	0.927 $\pm$ 0.024
T1568	1.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000	1.000 $\pm$ 0.000

Note: an initial pilot run of this experiment used only 10 trials per perturbation level; the increase to 100 trials (reported above and in Table 7.1c) was necessitated by the finding, discussed in Section 7.1.2, that single-attacker techniques (T1190, T1059) produce a binary

per-trial F1 outcome, requiring a larger trial count for a statistically meaningful mean/SD estimate.

B.3. Latency Reproducibility Diagnostic (Section 7.4)

The following supplementary measurements were taken during a diagnostic re-verification of the T1498 latency figure reported in Table 7.3, discussed in Section 7.4.

Table B.7: T1498 individual per-attacker latency measurements (diagnostic re-run, deepseek-r1:32b, all 70 attackers)

Statistic	Value
Number of individual measure_latency() calls	70
Mean total_ms across all 70 calls	4,601.4 ms
Minimum observed total_ms	3,954.27 ms
Maximum observed total_ms	16,299.89 ms

Table B.8: T1498 single-attacker latency, gpt-oss:20b (diagnostic re-run, first detected attacker only)

Statistic	Value
Measured total_ms	4,944.41 ms
Originally published value (Table 7.3)	1,440.60 ms
Absolute difference	3,503.81 ms

As discussed in Section 7.4, this divergence is attributed to transient server/network conditions at measurement time (e.g., model loading state, concurrent server load) rather than a change in pipeline logic, and is presented here as a transparent record of the reproducibility check performed.

Appendix C: Source Code Overview

This appendix provides an overview of the principal source files implementing the system described in Chapter 4 and Chapter 5, and the supplementary evaluation scripts underlying the experiments reported in Chapter 7. Full listings are provided in the accompanying project repository; this appendix summarises the structure and entry points of each file for reference.

C.1. mcp_server.py (MITRE ATT&CK MCP Server, Section 5.2)

- Framework instantiation: `mcp = FastMCP("MitreSecurityTool")`
- Data loading: `get_mitre_data()` — lazy-loads and caches enterprise-attack.json via `MitreAttackData`
- Helper functions: `safe_description(obj)`, `mitigation_object_from_wrapper(m)`
- Exposed tools:
 - `test_tool()` -> str
 - `get_technique_info(technique_id: str)` -> str
 - `get_balanced_mitigations(technique_id: str)` -> str
 - `full_security_analysis(technique_id: str)` -> str
- Transport: `mcp.run(transport="stdio")`

C.2. router_firewall.py (Firewall Enforcement MCP Server, Section 5.3)

- Framework instantiation: `app = FastMCP("RouterFirewall")`
- Configuration constants: `ROUTER_VPN_IP`, `SSH_USERNAME`, `SSH_KEY_PATH`
- State: module-level `blocked_ips: list`
- Helper function: `validate_ip(ip_string)` -> str — validates and canonicalises an address via Python's `ipaddress` module, raising `ValueError` for malformed or injection-bearing input (Section 8.4)
- Helper function: `ssh_to_router(command)` -> (success, output_or_error)
- Exposed tools:
 - `test_connection()` -> str
 - `block_ip(ip: str, technique_id: str = "")` -> str — calls `validate_ip(ip)` before command construction
 - `unblock_ip(ip: str)` -> str — calls `validate_ip(ip)` before command construction
 - `list_blocked()` -> str
 - `apply_mitre_mitigation(technique_id: str, attacker_ip: str)` -> str — calls `validate_ip(attacker_ip)` before command construction
 - `get_firewall_stats()` -> str
- Transport: `app.run()` (stdio)

C.3. evaluate_mcp_pipeline.py (Evaluation Pipeline, Section 5.4)

- Configuration constants: `BASE_DIR`, `NUM_RUNS = 30`, `DATASETS` (list of (pcap, csv, technique_id) tuples), `WEB_PORTS`, `REMOTE_PORTS`

- Feature extraction: `extract_features(pcap_file) -> list[dict]`
- Detection: `detect_mitre(f) -> str` (implements Table 4.1)
- Ground truth: `load_attackers(csv_path) -> set`
- Metrics: `evaluate(predicted, truth) -> dict`, `compute_statistics(values) -> dict`
- Baseline: `baseline_static_iptables(features, truth) -> dict`
- MCP invocation: `call_firewall(tool, args) -> str | None` (the MCP servers it connects to are registered in `mcp_router_config.json`, reproduced in Appendix A.6)
- Latency measurement: `measure_latency(technique_id, attacker_ip) -> dict`
- Per-dataset run: `run_single(pcap_path, csv_path, measure_lat=False) -> (dict, features, truth)`
- Full pipeline (30 runs): `run_pipeline(pcap_file, csv_file, mitre_id) -> dict`
- Scalability sweep: `run_scalability(pcap_file, csv_file, mitre_id) -> list[dict]`
- Summary printing: `print_summary(all_results)`
- Entry point: `if __name__ == "__main__":` — iterates over DATASETS, runs `run_pipeline` and `run_scalability` for each, writes `results_v3.json`

C.4. Supplementary Rule-Based Evaluation Scripts (Sections 7.1.1, 7.1.2)

Two additional standalone scripts were used to produce the cross-dataset validation (Table 7.1a/7.1b) and perturbation results (Table 7.1c), both reusing the `extract_features` and `detect_mitre` implementations from C.3 without modification:

- `cross_eval.py` — applies `detect_mitre` to every source across all five datasets simultaneously, using a `load_labels` function that maps each source IP to its ground-truth label and associated MITRE technique set (if any), and reports false positives by predicted technique (Table 7.1a/7.1b).
- `perturbation_experiment.py` — applies a `perturb_features(features, p_percent, seed)` function that multiplies `packet_count` and `avg_size` by $1 + U(-p, p)$ before re-running `detect_mitre`. Unlike an initial 10-trial pilot version of this script, the version used to produce the final results in Table 7.1c runs **100 independent trials** per perturbation level and records each trial’s F1 individually (`trial_f1s`), rather than only pooling TP/FP/FN across trials, so that a genuine mean and standard deviation across trials can be reported (Section 7.1.2). The increase from 10 to 100 trials was necessitated by a finding during development: for techniques with a single attacker source (T1190, T1059), each trial’s F1 is a binary outcome (0 or 1), and 10 trials proved too few to reliably estimate the true degradation rate — a methodological lesson discussed in Section 7.1.2.

C.5. LLM-Based Detection Scripts (Sections 7.6, 7.7)

The investigation into LLM-based detection without the rule-based filter (Section 7.6) and the MCP-retrieval experiments (Section 7.7) were implemented as a series of standalone

scripts, each corresponding to one experimental configuration. All scripts targeting the same dataset share the same underlying feature extraction and chunking logic as C.3, adapted for batch/relational-feature prompting where applicable.

Stage 1 (Section 7.6): - `stage1_llm_detection.py` — initial single-host baseline (Section 7.6.1) - `stage1_batch_llm_detection.py` — final selected batch/relational-feature configuration (Section 7.6.3); exposes `build_batch_prompt`, `extract_features`, `parse_batch_response`, reused directly by several Stage 3 scripts (C.6) to guarantee configuration consistency - `stage1_twobranch_llm_detection.py` — two-branch individual/group decomposition (Section 7.6.4) - `run_repeatability_robust.py` — 10-run repeatability harness with infrastructure-failure filtering (Section 7.6.5–7.6.6) - `stage2_validate_llm.py`, `stage2_validate_twobranch.py` — Stage 2 metric computation (Precision/Recall/F1, Micro/Macro averages) for the corresponding Stage 1 configurations - `error_analysis_t1190.py` — diagnostic re-run capturing explicit per-host reasoning text for the T1190 error analysis (Section 7.6.4)

Stage 3 (Section 7.7): - `experiment_3a_mcp_isolation.py` — single-host classification with agentic MCP access (Section 7.7.1) - `experiment_3b_mcp_best_config.py` — full-context configuration with agentic MCP access (Sections 7.7.2, 7.7.5, referred to as 3B/3F) - `experiment_3c_baseline_no_mcp.py` — full-context configuration with no MITRE information (Section 7.7.3) - `experiment_3d_hardcoded_no_mcp.py` — full-context configuration with engineered descriptions, no MCP (Section 7.7.4) - `experiment_3e_mcp_deterministic.py` — full-context configuration with MCP-retrieved descriptions injected deterministically (Section 7.7.4) - `experiment_3g_mcp_forced.py` — full-context configuration with forced agentic MCP tool use (Section 7.7.5) - `experiment_3h_compressed_mitre.py` — full-context configuration with a length-matched, compressed rewrite of the MCP-retrieved descriptions, isolating prompt length from semantic content (Section 7.7.6) - `validate_experiment.py` — generic Stage 2 validator accepting any experiment’s results file as a command-line argument; computes per-technique and aggregate metrics plus MCP-specific metrics (tool-call success rate, tool utilisation rate)

All Stage 3 scripts targeting agentic MCP use (3A, 3B/3F, 3G) bridge to the real MITRE MCP server (C.1) via the `mcp` Python client library over the `stdio` transport, rather than a simulated or re-implemented tool interface.

C.6. Patch Guidance: T1495 → T1498 Identifier Correction for Stage 3 MCP Queries

The dataset retains T1495 as a legacy internal label for artifact compatibility, but T1498 is the correct MITRE ATT&CK identifier for Network Denial of Service and is used for all corrected external MCP queries and written interpretation. As discussed in Sections 2.6.3, 6.2, and 7.2.2, the dataset files and internal code for the network-DoS technique retain the legacy identifier T1495, while the officially correct MITRE ATT&CK identifier is T1498 (Network Denial of Service; T1495 is the unrelated technique “Firmware Corruption”). The Stage 3 scripts that issue live queries to the MITRE MCP server for this technique — `experiment_3e_mcp_deterministic.py` and `experiment_3g_mcp_forced.py` (and, for its

classification-quality interpretation only, `experiment_3a_mcp_isolation.py`) — originally passed the dataset’s internal T1495 label directly as the `technique_id` argument to `mitre.get_technique_info` and related MCP tool calls, rather than remapping it to T1498 beforehand. This was the direct, identified cause of the construct-validity threat discussed in Section 7.2.2: for this technique, these experiments had retrieved and used MITRE content for T1495 (Firmware Corruption) rather than the intended T1498 (Network Denial of Service), in the results originally reported. `experiment_3h_compressed_mitre.py` issues no live MITRE MCP query at all — its technique descriptions are hard-coded in advance (Section C.3) — and is therefore unaffected by this identifier issue; it is discussed separately below.

Patch applied. Before any Stage 3 script constructs an MCP query, the following explicit identifier-mapping step was added:

```
# Map legacy internal dataset identifiers to their corrected,  
# officially valid MITRE ATT&CK identifiers before any external  
# MITRE MCP query is constructed. The underlying dataset files,  
# labels, and internal variable names are left unchanged; only the  
# identifier value passed to the MCP server is corrected.  
LEGACY_TECHNIQUE_ID_MAP = {"T1495": "T1498"}  
  
def resolve_mcp_technique_id(internal_technique_id: str) -> str:  
    return LEGACY_TECHNIQUE_ID_MAP.get(internal_technique_id,  
internal_technique_id)
```

This `resolve_mcp_technique_id` function was applied immediately before the technique identifier is passed to `mitre.get_technique_info` in each affected script’s MCP-query construction, and Experiments 3E and 3G were re-executed on the inference server following this correction; the corrected results are reported in Sections 7.2.2 and 7.7.4–7.7.8. Experiment 3H’s compressed technique descriptions were hard-coded in advance rather than retrieved live, and were confirmed on inspection to already convey the correct Network Denial of Service semantics; 3H’s script therefore required no code change, though it was re-executed once as an independent check, yielding a second, materially different Macro-F1 value discussed in Section 7.7.6. No other Stage 1, Stage 2, or Stage 3 script required this patch, since Experiments 3B/3F and 3C make no MITRE MCP calls for this technique’s classification, and 3D uses only the engineered, non-retrieved description already keyed to the correct semantic content.