network

MDPI

Article

Agentic AI Automation Framework for Computer Network Monitoring and Management

Alif Najwan bin Hishamuddin [1], Mohamed Hadi Habaebi [1,*], Md Rafiqul Islam [1] and Mohammed Abdullah Salem Al-Hussaini [2,*]

[1] Department of Electrical and Computer Engineering, Kulliyyah of Engineering, International Islamic University Malaysia, Jalan Gombak, Kuala Lumpur 53100, Malaysia; alif.najwan@student.iium.edu.my (A.N.b.H.); rafiq@iium.edu.my (M.R.I.)
[2] Faculty of Computer Studies, Arab Open University Oman, Muscat 130, Oman
* Correspondence: habaebi@iium.edu.my (M.H.H.); mohammed.h@aou.edu.om (M.A.S.A.-H.)

Abstract

Autonomous network management requires balancing rapid incident resolution with strict operational safety. This paper introduces a production-inspired, end-to-end multi-agent orchestration architecture for network fault diagnosis and resolution. Built upon an event-driven N8N orchestrator and integrated with live routing daemons, the framework couples the iterative Reasoning and Acting (ReAct) paradigm with topological Retrieval-Augmented Generation (RAG). To prevent unintended state disruptions, proposed remediation commands are evaluated using a dynamic Command Risk Score (R_C) and governed by a state-aware, two-way interactive Telegram Human-in-the-Loop (HITL) mechanism. Within the scope of six simulated fault scenarios evaluated on a three-router testbed, the multi-agent framework achieved high diagnostic precision and successful network restoration, significantly mitigating diagnostic hallucinations and reducing required human intervention. Furthermore, the system reduced Mean Time to Resolution (MTTR) compared to human engineering baselines. Furthermore, memory decay-based retrieval yields a 32.6% reduction in token consumption and a 34.4% operational cost saving per remediation event. This architecture demonstrates strong feasibility as a controlled proof-of-concept for automated diagnostic and remediation workflows within localized network environments.

Keywords: autonomous network management; closed-loop fault remediation; multi-agent LLM systems; ReAct reasoning architecture; Partially Observable Markov Decision Process (POMDP); Human-in-the-Loop (HITL) risk scoring; Retrieval-Augmented Generation (RAG); network telemetry & diagnostic accuracy

Check for updates

Academic Editor: Christos Bouras

Received: 11 September 2026
Revised: 28 September 2026
Accepted: 5 October 2026
Published: 8 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Network Operations (NetOps) has transitioned from manual, reactive management frameworks like FCAPS [1] to programmable, API-driven architectures [2]. Despite the development of rule-based Event-Condition-Action (ECA) automation, most operations remain at autonomy Levels 2 and 3, relying on human decision-making [3]. The primary challenges and proposed solutions in modern NetOps include:

- The inflexibility of rule-based ECA systems means they cannot handle novel faults [4]. As global connected devices are expected to exceed 125 billion by 2030 [5], operators face severe "alert fatigue," delaying fault resolution and increasing Mean Time to Resolution (MTTR).

- Large Language Models (LLMs) can bridge this reasoning gap by translating natural-language intent into executable actions and adapting to new fault manifestations without being limited by fixed rules [6].
- Deploying LLMs in production NetOps faces two major hurdles. First, there is a "last mile" disconnect where current LLMs act only as advisory layers without the ability to execute changes on live infrastructure [7]. Second, the well-known risk of AI hallucinations prevents the safe deployment of LLMs as fully autonomous, unsupervised actuators, as erroneous commands could cause severe network outages [8].

Intent-driven automation has emerged as a cornerstone paradigm for eliminating manual operational overhead in multi-vendor, NextG architectures. By translating high-level policy objectives into precise, deterministic device configurations, modern frameworks abstract low-level syntactic complexities while enabling zero-touch service provisioning [9]. Recent advances demonstrate that coupling Large Language Models (LLMs) with formal intent structures significantly enhances configuration fidelity across heterogeneous multi-vendor platforms, ensuring that high-level operational directives map accurately to underlying vendor-specific Command Line Interfaces (CLIs). Furthermore, integrating intent-driven control loops within softwarized paradigms, such as Open RAN, facilitates zero-touch service lifecycle management and dynamic resource orchestration across disaggregated radio controllers [10]. Synthesizing these intent-driven architectures into unified management loops, recent surveys emphasize that closed-loop zero-touch automation provides the necessary foundation for achieving Level 4 and Level 5 autonomous telecommunication operations.

The integration of agentic AI frameworks represents a paradigm shift from static, rule-based expert systems toward proactive, reasoning-driven diagnostic agents in network operations. By leveraging advanced LLMs, modern diagnostic agents can execute multi-step reasoning, ingest unstructured multi-modal telemetry, and dynamically select operational tools to diagnose complex network failures. Recent work shows that generative agent architectures operating within closed-loop assurance frameworks achieve high precision in telemetry ingestion, anomaly classification, and automated root-cause isolation [11]. To overcome the inherent constraints of stochastic LLM generation, such as hallucinated CLI commands or temporal memory loss, researchers have increasingly combined LLMs with structured domain knowledge graphs [12]. This hybrid agentic approach enhances diagnostic accuracy while retaining persistent contextual state across long-running troubleshooting sessions.

Artificial Intelligence for IT Operations (AIOps) provides the essential continuous-monitoring fabric required to detect subtle performance anomalies and execute autonomous self-healing routines. High-frequency telemetry streams often yield massive alarm volumes, leading to operator alert fatigue and prolonged Mean Time to Resolution (MTTR) [11]. Recent AIOps methodologies address this by applying AI-native recovery loops that automatically perform fault localization and trigger targeted remediation actions without human intervention. In cloud-native and virtualized telecommunication infrastructures, these automated self-healing mechanisms dynamically correlate disparate system metrics, map topology dependencies, and execute precision recovery protocols to contain service degradation before end-user quality of experience (QoE) is compromised.

Sustaining continuous operational assurance in softwarized and programmable network fabrics demands real-time, closed-loop telemetry and control pipelines. Traditional polling-based management protocols (such as SNMP) lack the granular temporal resolution required to capture sub-second transient faults or micro-burst traffic congestion. Modern closed-loop optimization architectures leverage gRPC/YANG-based streaming telemetry to feed continuous state updates directly into automated control agents, en-

abling dynamic performance tuning via deep reinforcement learning [13]. By combining low-latency telemetry ingestion with real-time feedback enforcement, these closed-loop paradigms provide comprehensive end-to-end visibility, enabling intelligent control loops to continuously optimize routing paths and adapt network configurations in response to dynamically fluctuating traffic conditions [14].

Figure 1 outlines the Proposed Agentic AI Network Monitoring and Management System, tested on a virtualized FRRouting (FRR)/OSPF multi-router testbed. The three-layer architecture (Infrastructure, Orchestration, and Intelligence) uses a ReAct reasoning pipeline with six agents: Master Orchestrator, Perception, Memory, Thought Process, Decision Maker, and Human-in-the-Loop (HITL) Executor. Unlike purely advisory LLM systems, it interfaces directly with actuators via SSH on live devices. To prevent LLM hallucinations, a two-way Telegram HITL safety gate allows operators to accept or reject AI-proposed actions. Additionally, a retrieval-augmented continuous JSON memory enables the system to reuse past solutions without retraining GPT-4o. The architecture was evaluated across 30 trials for six fault types from four OSI layers, comparing success rates and MTTR against developer and blind-evaluator baselines.

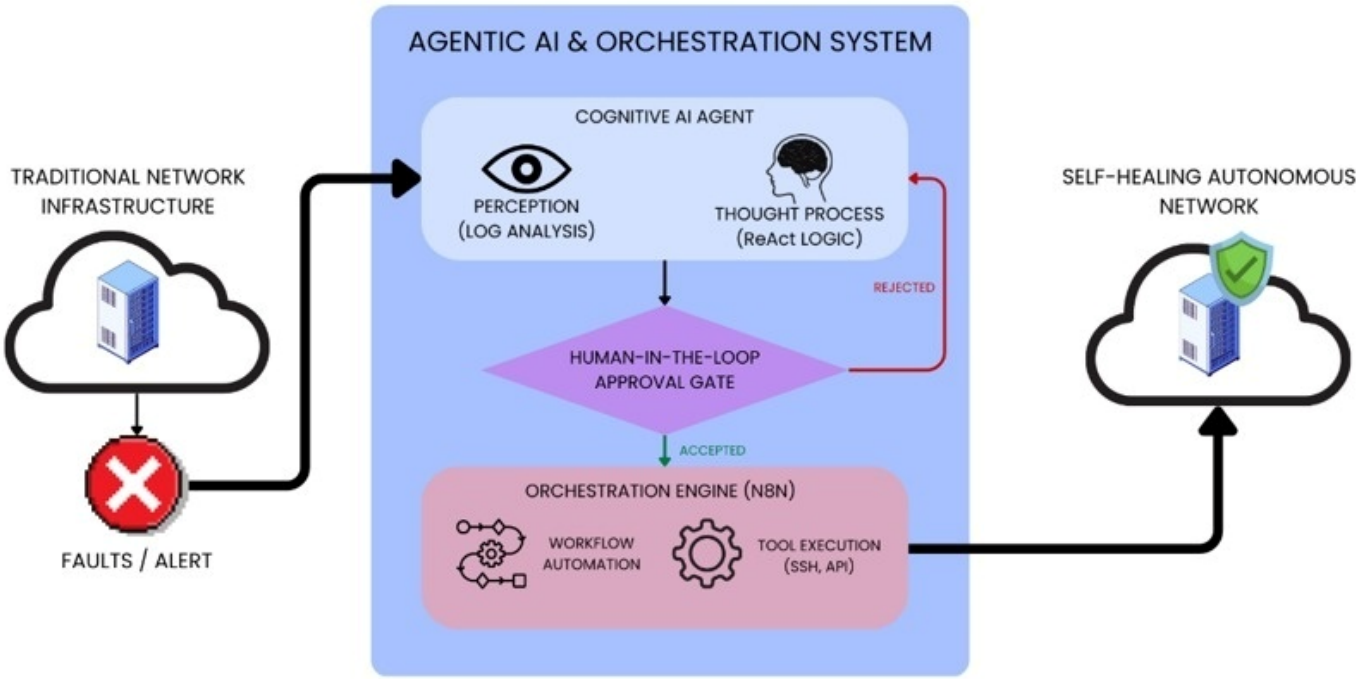

Figure 1. Overview of the Proposed Agentic AI Network Monitoring and Management System.

Classical network operations rely on the FCAPS (Faults, Configurations, Accounting, Performance, Security) model, which is fundamentally reactive and human-driven. Detecting faults requires manually correlating logs and telemetry from different tools, a laborious process highly susceptible to human error as network scale increases [1]. Early automated diagnostics reduced effort via rule-based inference, but the introduction of SDN and RESTful APIs marked a breakthrough by decoupling control and data planes for programmatic management [2]. This enabled scripted Event-Condition-Action (ECA) automation. However, ECA solutions cannot reason through unexpected faults, leading to human escalation. Academia defines autonomy across TM Forum's six levels (Table 1). Most current production systems remain at Levels L2 and L3, where systems execute and humans decide, while fully automated L4 autonomy remains an open research problem [3].

Table 1. TM Forum Autonomous Network Levels.

Level	Classification	Decision Making
L0	Manual	Human
L1	Assisted	Human
L2	Partial	Human
L3	Conditional	Human/System
L4	High Autonomous	System
L5	Fully Autonomous	System

LLMs offer a semantic bridge between high-level operator intent and low-level configuration syntax, overcoming the reasoning limits of rule-based automation [6,15]. Agentic approaches in Software-Defined Networks show LLMs can accurately translate operator intent into corresponding SDN control actions, illustrating the feasibility of natural-language-mediated network configuration [6], while model-driven orchestration can guide distributed infrastructures using intents rather than manual scripts [15]. Retrieval-Augmented Generation (RAG) further grounds LLM outputs in domain context to improve reliability [7]. However, most frameworks use LLMs strictly as advisory layers [6,15], lacking direct actuation [7] and requiring manual deployment. Furthermore, LLMs' tendency to hallucinate [16] necessitates a validated execution and safety layer before applying changes to physical networks.

Low-code orchestration platforms provide a middleware bridge between LLM reasoning and executable automation [16–21]. N8N's API-first architecture facilitates rapid integration of agentic pipelines across siloed toolchains without extensive custom development [22]. In large-scale operations, combining ChatOps interfaces with ReAct prompting improves LLM correctness on multi-step diagnostic tasks compared to single-shot approaches. However, few studies implement ReAct reasoning with direct, automated SSH actuation on live infrastructure [17]. This gap directly informs our proposed six-agent N8N pipeline on an FRRouting/OSPF testbed.

Agentic AI architectures are increasingly applied to broader infrastructure operations, such as multi-agent cloud incident troubleshooting [5] and AI data center maintenance [3]. For Beyond-5G/6G networks, meeting strict latency and closed-loop control requirements remains challenging [4]. While LLM-driven agents for autonomous remediation are conceptually feasible [23], empirical validation on physically instantiated routing infrastructure remains rare, a gap addressed by this study's testbed.

Human-in-the-Loop mechanisms are critical safeguards against autonomous AI risks [8]. Approval gates mitigate hallucination-induced outages while maintaining operational flexibility for routine tasks. However, existing literature lacks empirical validation of bidirectional HITL systems [24]. This study fills this gap by allowing operators to actively reject, reroute, and halt pipeline execution, demonstrated via a Neighbor Down OSPF fault scenario.

Existing LLM-driven network management systems exhibit three structural deficiencies: they operate in an open-loop paradigm without live telemetry verification, rely on unidirectional binary HITL gates instead of bidirectional dynamic risk scoring, and lack validation on physical network testbeds running production routing protocols. Table 2 summarizes these gaps across state-of-the-art architectures, explicitly contrasting them with this paper's contributions.

Recent advancements in LLM-driven network automation have demonstrated the potential of generative AI for log parsing, telemetry analysis, and automated script generation. However, existing approaches generally suffer from three core architectural deficiencies:

1. Open-Loop Execution: Most existing frameworks operate in an open-loop fashion, generating CLI commands or diagnostic suggestions without verifying state convergence on live network infrastructure.
2. Coarse Safety Policies: Current systems rely on static command blocklists rather than dynamic contextual risk scoring, making them incapable of differentiating between benign operational queries and high-impact actions during live troubleshooting.
3. Passive Human Involvement: Human-in-the-Loop (HITL) implementations in literature typically function as out-of-band audit mechanisms rather than real-time, interactive gatekeepers.

Table 2. Comparison of Prior Agentic AI and Automation Approaches in Network Management.

Ref.	Platform	Use Case	AI Lvl	Adapt.	HITL Support	Physical Deployment	Closed-Loop Verification	Memory Augmentation
[18]	Python LLM Agents	Wireless QoS Opt.	L4	High	Partial	Simulated	No	Basic
[17]	N8N + LangGraph + ChatOps	Troubleshoot /Provision	L4	High	Yes	Virtual	No	Session
[19]	LangChain + R-BAI	NMS Diagnosis	L3	Med	No	Virtual	No	Vector Store
[20]	Fine-tuned GPT-2	Threat Detection	L2	Low	No	Offline/Log	No	None
[16]	N8N + LangGraph + CML	Auto. Mgmt. /Remed.	L4	High	Partial	Virtual (CML)	Basic	Short-term
[21]	GAI Digital Twin	Closed-Loop Mgmt.	L5	High	Optional	Digital Twin	Yes (Simulated)	Episodic
This work	N8N + GPT-4o	Fault Remediation	L4	High	Explicit (Interrupt /Approval)	Hardware /Multi-Router	Yes (Post-Fix Telemetry Check)	Structured State Memory

To bridge these gaps, this work introduces a production-inspired, closed-loop multi-agent orchestration architecture. Unlike standard LLM script generators or static automation workflows, our specific contributions include:

- Closed-Loop N8N/ReAct Orchestration: A fully automated, closed-loop pipeline built on N8N that couples the iterative Reasoning and Acting (ReAct) paradigm with active routing daemon telemetry (FRRouting/BGP/OSPF), ensuring that every diagnostic iteration and remediation command is validated against live control-plane states.
- Dynamic Command Risk Scoring (R_C): A quantitative, contextual risk-assessment engine that evaluates candidate CLI actions in real time prior to execution, scoring commands based on parameter severity, blast radius, and target node criticality.
- Two-Way Interactive HITL Gatekeeping: An asynchronous, state-aware Telegram approval/rejection mechanism integrated directly into the live execution stream. This enables human operators to review proposed actions, inspect dynamic risk scores (R_C), and approve or abort state modifications in real time before commands are dispatched to live routing daemons.

The paper is organized as follows: Section 2 details the Design methodology, architecture, agents, ReAct prompting, HITL gate, and testbed. Section 3 introduces the performance metrics and the evaluation approach. Section 4 discusses experimental results,

memory effects, and HITL rejection scenarios. Section 5 covers the limitations of the study and future work, and Section 6 concludes the paper.

2. Materials and Methods

2.1. Research Approach & Mathematical Abstraction

This research adopts the Design Science Research (DSR) framework as its guiding methodology, structured across four sequential phases: (1) Requirement Analysis and Tool Selection, (2) System Architecture Design, (3) Prototype Implementation, and (4) Experimental Evaluation. DSR was selected in preference to purely descriptive or exploratory research designs because the central aim of this study is not merely to characterize an existing phenomenon, but to construct and empirically validate a novel artifact, namely the six-agent agentic AI pipeline, whose utility can only be established through iterative design, functional implementation, and controlled experimentation against a real operational baseline. Phase 1 involved a systematic review of existing NetOps automation approaches and the selection of N8N v.2.40.6 (n8n GmbH, Berlin, Germany) and GPT-4o (API available through gpt-4o-2024-05-13 snapshot, OpenAI, San Fransesco, CA, USA) as the core orchestration and reasoning components, on the basis of N8N's node-based, API-first extensibility and GPT-4o's demonstrated capacity for multi-step reasoning over natural language and structured diagnostic output. Phase 2 encompassed the design of the modular three-tier system architecture, the specification of ReAct-based reasoning prompts for each agent, and the definition of the bidirectional HITL safety gate. Phase 3 comprised the physical implementation of the virtualized testbed and the integration of the N8N pipeline with the testbed via webhooks and REST APIs. Phase 4 consisted of the controlled experimental evaluation described in Section 4, in which the completed prototype was benchmarked against manual troubleshooting baselines across 30 trials. This four-phase structure, illustrated in Figure 2, ensured that each architectural and implementation decision was traceable back to a requirement identified during the literature review, and that the resulting artifact was subjected to a rigorous, quantifiable evaluation rather than a purely qualitative demonstration.

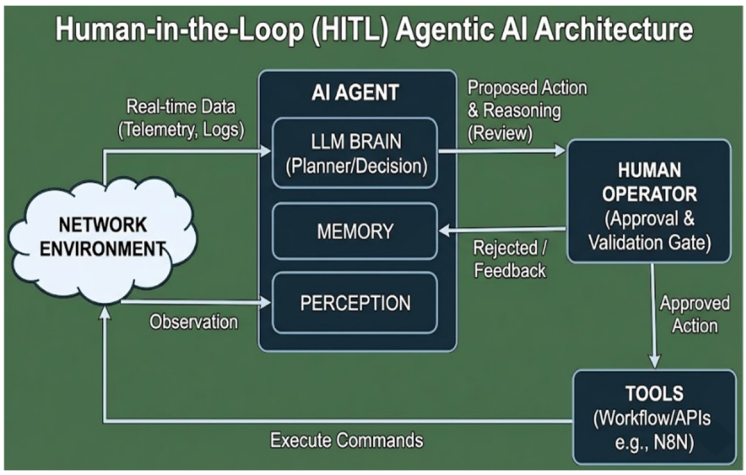

Figure 2. The Human-in-the-Loop (HITL) Architecture Integrating the OODA Control Cycle.

To ground our autonomous troubleshooting pipeline in rigorous control theory, we formulate the network diagnostic process as a Partially Observable Markov Decision Process (POMDP). Within this framework, the underlying global network state $s_t \in \mathcal{S}$ is not directly visible to the agent due to telemetry masking, missing logs, or silent failure modes. Instead, the agent interacts with the system by issuing diagnostic observation

commands $a_t \in \mathcal{A}$ (e.g., CLI show commands) to collect partial observations $o_t \in \mathcal{O}$ (e.g., routing tables, interface status, syslog outputs).

It is important to clarify that this POMDP formulation serves as a formal decision-theoretic abstraction mapping directly onto our iterative ReAct (Reasoning + Acting) loop. Rather than computing explicit high-dimensional belief state matrices over the full network state space, the operational N8N workflow engine implements this abstraction implicitly. The agent's cumulative prompt history and structured memory buffer maintain the belief state $b(s_t)$, while the LLM engine acts as the policy network $\pi(a_t|b(s_t))$, selecting the next diagnostic action based on prior observations to incrementally reduce topological uncertainty.

To establish a rigorous mathematical foundation for autonomous closed-loop NetOps, we formulate the network domain model, the event state space, the decision transformation, and the network fault remediation as a discrete state-space transformation problem.

Network Domain Model: Let the physical network infrastructure be defined as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}_{\text{link}})$, where $\mathcal{V}$ denotes the set of virtualized routing nodes ($|\mathcal{V}| = 3$) running FRRouting/OSPF and $\mathcal{E}_{\text{link}}$ represents active inter-router link interfaces. The baseline network operational state at time t is represented by a high-dimensional vector $s_t \in \mathcal{S}$, encapsulating interface states, routing table entries, and protocol adjacency metrics.

Event State Space: An operational anomaly or fault injection induces a state transition from the nominal baseline $s_0 \in \mathcal{S}$ to a degraded state $s_t \neq s_0$. This degradation triggers an alert event $e_t \in \mathcal{E}$, defined as a tuple:

$$e_t = \langle t_0, v_i, \mathrm{m}_t \rangle \tag{1}$$

where t_0 is the incident timestamp, $v_i \in \mathcal{V}$ is the originating node, and m_t represents telemetry log payloads (e.g., syslog entries, SNMP traps, or failed ping signatures).

Decision Transformation: The objective of automated network management is to derive an optimal decision policy $\Psi : \mathcal{E} \times \mathcal{M} \to \mathcal{C}$ that maps an anomalous event e_t and historical context memory $\mathcal{M}$ into a deterministic sequence of corrective CLI commands $\mathrm{c} = (c_1, c_2, \ldots, c_m) \in \mathcal{C}$. Execution of c acts as a state transition operator:

$$s_{t+\Delta t} = T(s_t, \mathrm{c}) \tag{2}$$

such that $s_{t+\Delta t} \to s_0$, fully restoring nominal routing convergence and data-plane reachability within the mean time to resolution $\Delta t = T_{\text{MTTR}}$. This decision transformation Ψ is operationalized via the proposed multi-agent framework.

Network Fault Remediation: To account for incomplete telemetry logs and dynamic state changes, we formalize the multi-agent diagnostic and remediation process as a Partially Observable Markov Decision Process (POMDP) defined by the tuple

$$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, \mathcal{O}, \gamma)$$

where

$\mathcal{S}$: True network state space $s_t \in \mathcal{S}$ (including hidden link/daemon failures).
Ω : Telemetry observation space $\omega_t \in \Omega$ (syslog payloads, ICMP status, CLI outputs).
$\mathcal{O}(\omega_t \mid s_t, a_{t-1})$: Observation probability function mapping true state s_t to diagnostic logs.
$\mathcal{A} = \mathcal{A}_{\text{diag}} \cup \mathcal{A}_{\text{rem}}$: Action space partitioned into diagnostic queries and corrective CLI executions.

2.2. System Architecture Overview

The proposed framework is organized according to a modular three-tier architecture comprising an Infrastructure Layer, an Orchestration (Middleware) Layer, and an Intelligence Layer [20,21], as depicted in Figure 3. This layered decomposition was adopted

deliberately to decouple the concerns of physical network actuation, workflow control logic, and cognitive reasoning, such that any individual layer could be modified, replaced, or scaled independently without necessitating redesign of the other two. The Infrastructure Layer is responsible for hosting the live network devices, detecting fault events, and executing remediation commands. The Orchestration Layer, implemented as a self-hosted N8N instance, is responsible for coordinating the sequence of diagnostic and decision-making steps, managing state through a persistent memory store, and enforcing the human approval gate prior to any state-changing action. The Intelligence Layer, powered by GPT-4o accessed via the OpenAI API, supplies the cognitive reasoning capability that each of the six agents draws upon to interpret fault symptoms and generate remediation plans. This separation of concerns directly addresses the research gap identified in Section 1, in which prior LLM-network management integrations positioned the model as an isolated advisory layer; here, the Intelligence Layer is embedded within, and structurally inseparable from, an Orchestration Layer that possesses direct actuation authority over the Infrastructure Layer, closing the loop between reasoning and execution.

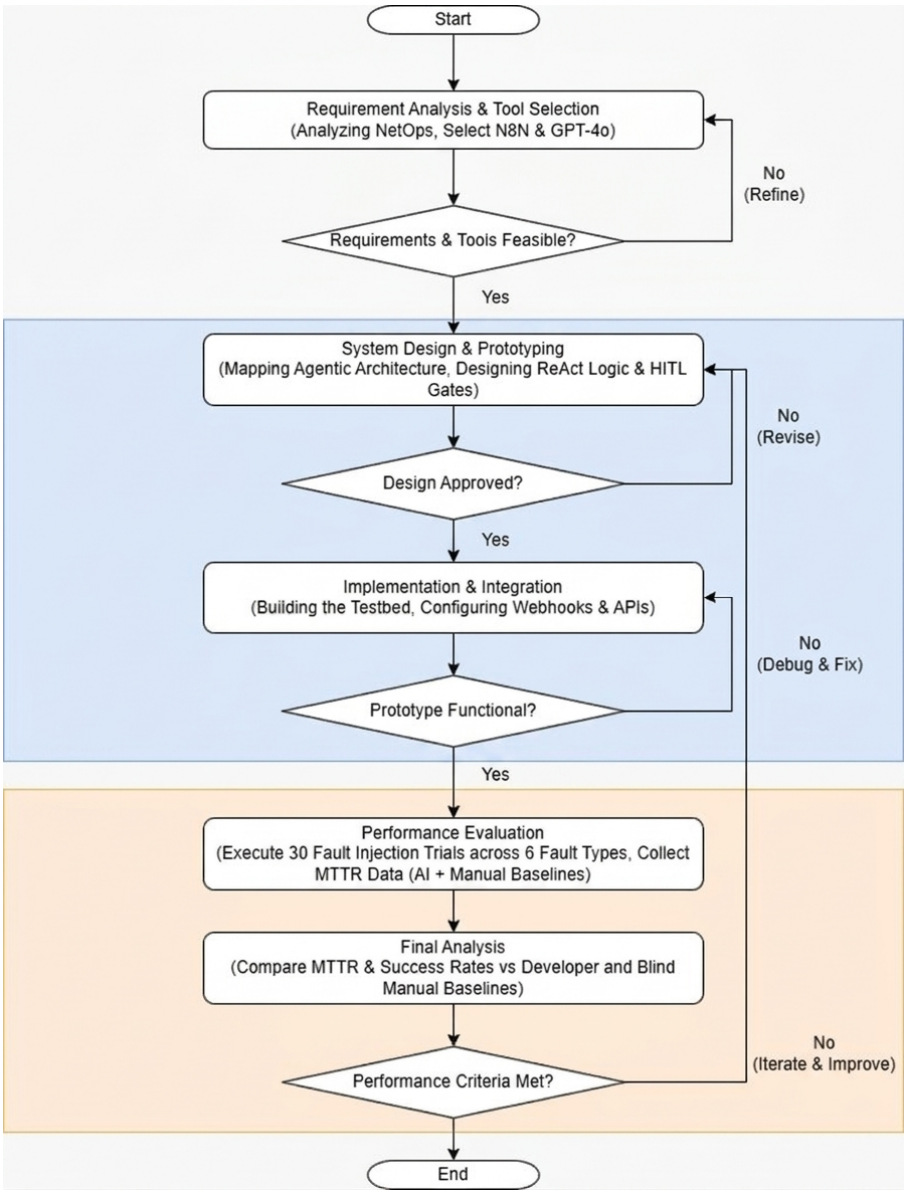

Figure 3. Research Methodology Flowchart.

Formally, the physical topology is modeled as a graph $G = (V, E)$, where V represents the set of FRRouting router nodes ($|V| = 3$) and E denotes active data-plane links. The operational network state at time t is defined as $S_t \in \mathcal{S}$. Upon detecting an operational anomaly $e \in \mathcal{E}$, the overall multi-agent pipeline operates as a composite transformation function $\Phi : \mathcal{E} \times \mathcal{M} \to \mathcal{C}$ mapping the event e and historical context $\mathcal{M}$ to an executed remediation command vector $\mathbf{c} \in \mathcal{C}$:

$$\Phi(e, \mathcal{M}) = (A_6 \circ A_3 \circ A_{2B} \circ A_{2A} \circ A_1 \circ A_0)(e, \mathcal{M}) \quad (3)$$

2.3. Infrastructure Layer

The network topology, architecture, and operational cycle of the agentic AI with the ReAct framework is shown in Figure 4. The Infrastructure Layer consists of three virtual machines running Ubuntu 22.04 LTS, provisioned using VMware Workstation Pro, with each virtual machine hosting an instance of FRRouting (FRR) configured to run the Open Shortest Path First (OSPF) protocol, collectively forming a three-router topology that mimics a small-scale enterprise or campus network. The three routers were assigned OSPF router identifiers of 1.1.1.1, 2.2.2.2, and 3.3.3.3, and were interconnected over a dedicated OSPF data-plane network (ospf-net, 10.0.0.0/24) on which OSPF Hello and Link-State Advertisement (LSA) traffic was exchanged to establish and maintain neighbor adjacencies. A separate management-plane network (VMnet1, 192.168.56.0/24) was used to isolate administrative and monitoring traffic from the OSPF data plane, ensuring that syslog transmission and SSH-based diagnostic access did not interfere with, or become confounded by, the routing traffic under observation. Table 3 summarizes the testbed's virtual machine and network configuration. Fault detection within this layer is performed through two complementary mechanisms. First, a Python-based syslog bridge listens for live fault events emitted by the FRR daemons on each router and forwards them to the Orchestration Layer via HTTP webhooks, providing an event-driven detection pathway for faults that generate an explicit syslog signature, such as OSPF adjacency state changes. Second, a Paramiko-based connectivity check agent actively and continuously probes the network using Internet Control Message Protocol (ICMP) reachability tests, independent of whether a corresponding syslog event has been generated; this active-probing mechanism ensures that faults which do not produce a distinctive syslog signature, such as a silently configured routing blackhole, are nonetheless detected. Remediation commands generated by the Intelligence Layer and approved through the HITL gate are executed against the routers via SSH using the Paramiko library, completing the actuation pathway from diagnosis to physical network state change.

Table 3. Testbed Virtual Machine and Network Configuration.

Component	Configuration
Host Platform	VMware Workstation Pro
Guest OS	Ubuntu 22.04 LTS (×3)
Routing Software	FRRouting (FRR) with OSPF
Router IDs	1.1.1.1, 2.2.2.2, 3.3.3.3
OSPF Data Plane	ospf-net, 10.0.0.0/24
Management Plane	VMnet1, 192.168.56.0/24
Fault Detection	Python syslog bridge 3.20 + Paramiko ICMP probe
Remediation Actuation	SSH via Paramiko

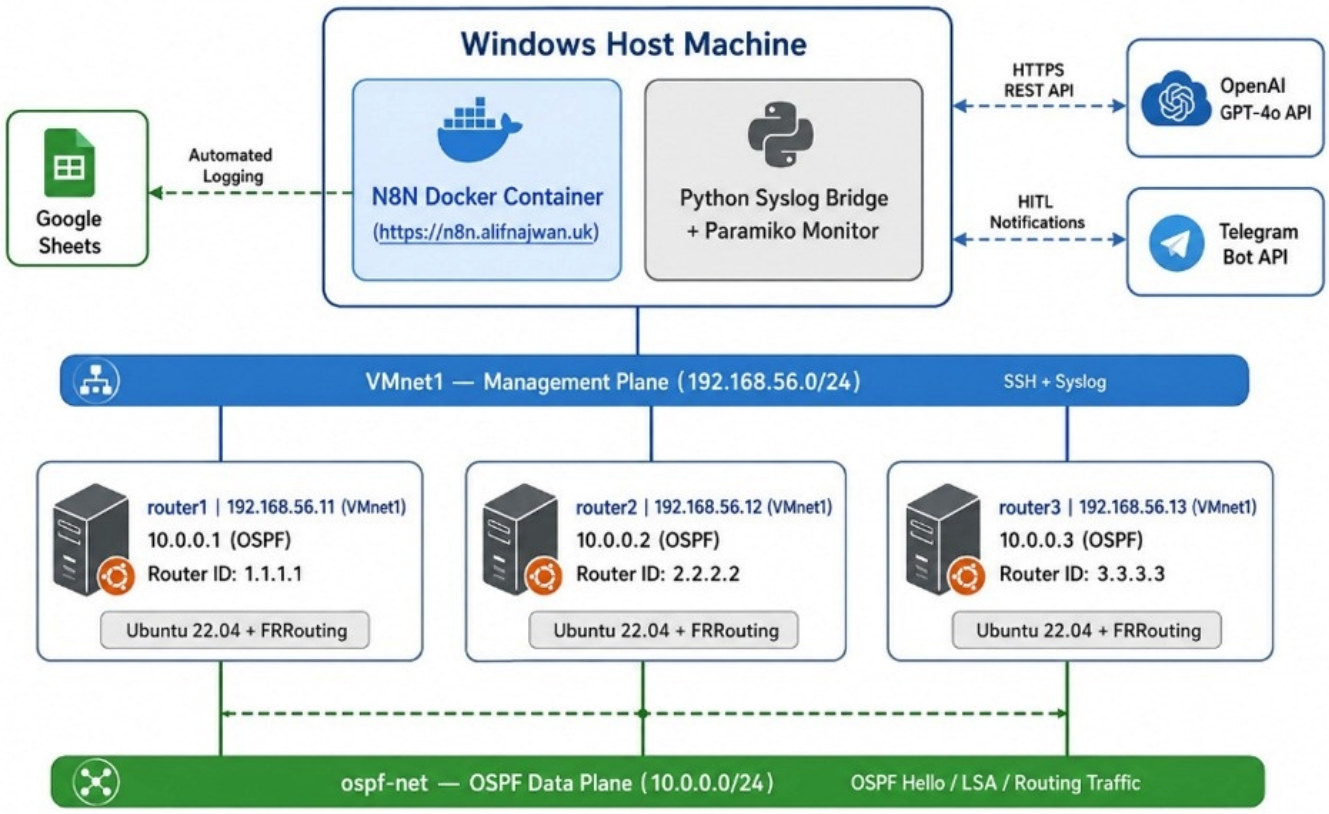

Figure 4. Network Topology, Architecture, and Operational Cycle of the Agentic AI with the ReAct Framework.

2.4. Orchestration Layer and Agent Design

The Orchestration Layer is implemented as a self-hosted N8N workflow automation engine, within which the agentic reasoning pipeline is structured as six discrete, sequentially coordinated agents [20]. This decomposition follows the ReAct (Reason + Act) paradigm, in which each agent performs a bounded reasoning step, invokes an external action such as an SSH diagnostic command, observes the result, and passes an updated context forward to the next agent in the chain, rather than attempting to resolve the entire fault in a single monolithic reasoning pass. The six agents are described individually below, with the complete pipeline flow illustrated in Figure 5.

(1) Master Orchestrator: The Master Orchestrator serves as the entry point of the pipeline, receiving incoming fault events from the Infrastructure Layer's syslog bridge and connectivity monitor via webhook. Upon receipt of an event, it performs initial alert normalization and severity classification, determining whether the event warrants initiation of the full diagnostic pipeline or can be discarded as non-actionable noise, before triggering the downstream agents.

(2) Agent 1 (Perception): Agent 1 performs the initial classification of the fault, drawing on memory-based pattern matching against previously recorded incident signatures and issuing an initial round of SSH-based diagnostic commands to confirm the affected device and fault category. This agent corresponds to the first "Reason" step in the ReAct loop, narrowing the space of candidate fault types before deeper diagnosis is undertaken.

(3) Agent 2A (Memory): Agent 2A queries the persistent JSON memory store for prior incidents matching the fault signature identified by Agent 1. Where a sufficiently similar prior resolution exists, this agent retrieves the associated diagnostic and

remediation trace, enabling the pipeline to bypass redundant multi-step reasoning in subsequent stages, an effect that is examined empirically in Section 4.

(4) Agent 2B (Thought Process): Agent 2B constitutes the core ReAct reasoning loop of the pipeline, conducting in-depth diagnosis through iterative rounds of SSH-based command-line interface (CLI) checks, including inspection of the routing table, interface status, and service status of the affected device. Where the fault signature is ambiguous, as is the case for the four fault types sharing the AdjChg syslog signature, Agent 2B issues successive rounds of diagnostic commands, reasoning over each result before determining whether sufficient information has been gathered to proceed to remediation planning.

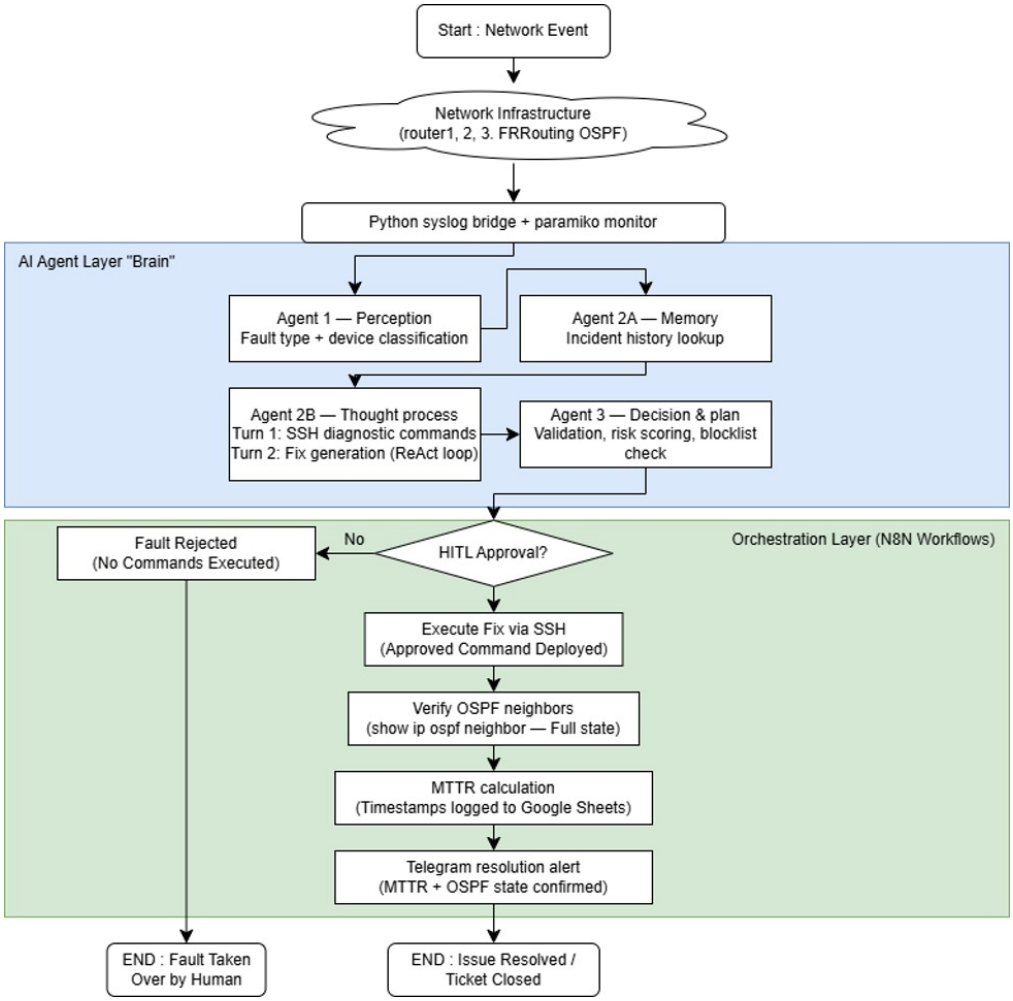

Figure 5. N8N Multi-Agent Pipeline Flowchart.

Mathematically, at diagnostic iteration $k \in \{1, \ldots, K\}$, Agent 2B generates a state tuple $\tau_k = (T_k, a_k, o_k)$, where T_k represents the internal cognitive thought, $a_k \in \mathcal{A}_{\text{diag}}$ is an SSH diagnostic command, and $o_k = \text{Exec}(a_k, S_t)$ is the observed CLI response. The thought state T_k is updated recursively according to:

$$T_k = f_{\text{LLM}}(P_{\text{ReAct}}, \tau_1, \tau_2, \ldots, \tau_{k-1}, e) \tag{4}$$

where f_{LLM} denotes the GPT-4o inference function conditioned on the ReAct system prompt P_{ReAct} and accumulated observation trajectory.

At step k, Agent 2B updates its belief state over potential root causes based on the accumulated observation trajectory τ_k, selecting the optimal diagnostic action a_{k+1} to

maximize information gain. Model Agent 2B's ReAct diagnostic loop as a belief state update $b_k(s_t) = P(s_t \mid \tau_k)$, where observation trajectory $\tau_k = (\omega_0, a_1, \omega_1, \ldots, a_k, \omega_k)$:

$$\tau_k = \tau_{k-1} \parallel (a_k, \text{Exec}(a_k, s_t)) \tag{5}$$

$$a_{k+1} = \text{argmax}_{a \in \mathcal{A}_{\text{diag}}} \mathbb{E}[H(b_k) - H(b_{k+1}) \mid a] \tag{6}$$

(where $H(b)$ represents entropy over candidate root causes, formalizing information gain per diagnostic command).

(5) Agent 3 (Decision Maker): Agent 3 synthesizes the diagnostic findings produced by Agents 1, 2A, and 2B into a concrete remediation plan, generating the specific CLI commands required to restore normal network operation. This agent additionally performs a risk-scoring step and a blocklist check, cross-referencing the proposed commands against a predefined list of high-risk operations, before the plan is forwarded to the HITL Executor for human review.

To quantify execution risk prior to human dispatch, Agent 3 evaluates a scalar risk metric $R(\mathbf{C}) \in [0, 1]$ over the proposed remediation sequence $\mathbf{C} = (c_1, c_2, \ldots, c_m)$:

$$R(\mathbf{C}) = 1 - \prod_{j=1}^{m} \left(1 - \sigma\left(\mathbf{w}^T \mathbf{x}(c_j)\right)\right) \tag{7}$$

where $\mathbf{x}(c_j)$ is a binary feature vector denoting command impact level (e.g., interface status change vs. global route flushing), $\mathbf{w}$ is an empirically assigned weight vector, and $\sigma(\cdot)$ is the standard logistic function. If $R(\mathbf{C}) \geq \theta_{\text{risk}}$, mandatory human sign-off via the HITL gate is automatically enforced.

(6) HITL Executor: The HITL Executor governs the interface between AI-generated decisions and physical network actuation. It transmits the proposed diagnosis and remediation commands to the operator via a Telegram bot for approval, and, contingent on the operator's response, either executes the approved commands via SSH or halts the pipeline entirely. The design and empirical validation of this bidirectional gate are described in detail in Section 2.

During diagnostic exploration, the agent prioritizes actions that maximize information gain regarding the true underlying network fault. The uncertainty of the belief state is modeled via Shannon entropy:

$$H(b_t) = -\sum_{s \in \mathcal{S}} b_t(s) \log_2 b_t(s)$$

An optimal diagnostic command a_t^* aims to maximize the expected entropy reduction (information gain $I(a_t)$) across potential telemetry observations:

$$I(a_t) = H(b_t) - \mathbb{E}_{o \sim P(o|b_t, a_t)}[H(b_{t+1}(s|b_t, a_t, o))]$$

In our concrete implementation, these entropy-reduction equations act as a theoretical bridge connecting decision-theoretic modeling with concrete N8N workflow execution. Rather than executing numerical matrix updates for $H(b_t)$ at runtime, the LLM-driven agent leverages this optimization objective within its ReAct prompt template. The model is instructed to evaluate candidate diagnostic commands by weighing their potential to eliminate competing hypothesis branches against their execution cost and risk. This enables the N8N orchestrator to dynamically prune irrelevant CLI queries and rapidly converge on the true root cause through targeted telemetry probing.

2.5. Intelligence Layer and ReAct Prompt Engineering

The Intelligence Layer employs GPT-4o, accessed via the OpenAI API, as the shared cognitive reasoning engine underlying all six agents. Each agent is instantiated as a distinct prompt context within the N8N pipeline, rather than as a separately trained model, such that agent specialization is achieved entirely through prompt engineering and scoped context rather than through model fine-tuning. This design choice was made deliberately to preserve the system's adaptability: because no agent-specific fine-tuning is performed, the memory-store learning effect described in Section 4 is achieved through a purely retrieval-augmented mechanism, allowing the pipeline to improve its efficiency on recurring fault types without incurring the cost or latency of retraining the underlying model. Each agent's prompt was constructed to explicit reflect the ReAct structure, requiring the model to articulate an intermediate reasoning trace (the "Thought") before specifying the corresponding diagnostic or remediation action (the "Action"), and to incorporate the result of that action (the "Observation") into its subsequent reasoning step. This explicit interleaving of reasoning and action was intended to reduce the incidence of hallucinated or unsupported remediation commands, since each proposed action is grounded in an observation obtained directly from the live network state rather than generated purely from the model's prior knowledge.

2.6. Human-in-the-Loop Design

The HITL gate is implemented as a bidirectional approval mechanism interposed between Agent 3's remediation plan generation and the physical execution of that plan by the HITL Executor. Prior to any state-changing action being applied to the network, the HITL Executor transmits an approval request to the operator via a Telegram bot, presenting both the fault diagnosis produced by the upstream agents and the specific remediation commands proposed, together with their predicted effect on network state. The operator then afforded two possible responses. If the operator approves the proposed action, the HITL Executor proceeds to execute the specified commands against the target router via SSH, and the pipeline subsequently verifies restoration of OSPF adjacency and end-to-end connectivity before logging the trial outcome. If the operator instead disapproves the recommendation, the pipeline terminates execution immediately, no commands are deployed to the network, and the system awaits a revised instruction from the operator before any further action is taken. This design was chosen specifically to address the gap identified in Section 1, in which prior HITL implementations in the literature were validated only in their approval pathway; the rejection pathway is empirically exercised and confirmed functional in Section 4.

2.7. Fault Scenarios and Injection Procedures

Six fault scenarios were designed to span four distinct OSI layers and a range of diagnostic complexity profiles, as detailed in Table 4. Each fault was injected manually via CLI on the designated router virtual machine, with injection commands selected to reproduce realistic, operator-induced or naturally occurring failure conditions rather than artificial or contrived error states. The OSPF Neighbour Down, iptables ACL Block, Interface Admin Down, and Route Blackhole fault types share a common ambiguous syslog signature (AdjChg), such that Agent 2B is required to perform active, multi-step SSH-based diagnosis to disambiguate the underlying cause; this shared-signature design was deliberate, intended to test the pipeline's diagnostic reasoning capability under conditions of genuine classification ambiguity rather than trivial pattern matching. By contrast, the Timer Mismatch fault produces a unique HelloInterval syslog signature, permitting direct classification without extended multi-step reasoning, and thereby serving as a lower-complexity reference point against which the diagnostic overhead incurred by the ambiguous-signature faults could

be measured. Remediation was considered successful only when full OSPF adjacency and end-to-end connectivity were restored and independently confirmed by the Paramiko connectivity monitor, rather than relying solely on the absence of further syslog alerts.

Table 4. Fault Scenario Definitions and Injection Methods.

Fault Type	OSI Layer	Detection Method	Injection Method
OSPF Neighbour Down	Layer 3	Syslog (AdjChg)	OSPF area misconfiguration
iptables ACL Block	Layer 3/4	Syslog + ICMP probe	iptables -A INPUT -j DROP
Interface Admin Down	Layer 2	Syslog + ICMP probe	ip link set eth1 down
Route Blackhole	Layer 3	ICMP probe + route check	ip route add blackhole
FRR Service Crash	Layer 7	Syslog + service check	systemctl stop frr
Timer Mismatch	Layer 3	Syslog (HelloInterval)	OSPF hello-interval mismatch

2.8. Comparative Architectural Position

To explicitly differentiate our platform from state-of-the-art LLM-based network management frameworks, Table 5 compares key architectural dimensions.

Table 5. Feature comparison with existing LLM-driven network management approaches.

Feature/Dimension	Traditional Automation (Ansible/Nornir)	Existing LLM Frameworks (e.g., NetAgent)	Proposed Framework
Execution Loop	Static/Open-Loop	Open-Loop/Semi-Automated	Closed-Loop (ReAct + Verification)
Live Daemon Integration	Scripted CLI/API	Passive Log Parsing	Direct Interactive Routing Daemons
Safety Mechanisms	Static Rule Sets	Basic Blocklists	Dynamic Contextual Risk Score (R_C)
HITL Governance	Manual Playbook Run	Static Confirmation	Two-Way Real-Time Telegram Gatekeeper
Memory Augmentation	None	Vector RAG Only	Topological RAG + Episodic Memory

As detailed in Table 5, the distinguishing strength of our platform lies in the tight coupling of the N8N event-driven orchestrator with dynamic risk scoring (R_C) and interactive two-way HITL controls. While prior systems generate static remediation proposals, our pipeline executes a closed-loop ReAct sequence: every proposed state modification on live routing daemons is dynamically risk-evaluated. If the calculated command risk score R_C exceeds safe threshold bounds, execution pauses automatically, dispatching an interactive prompt via Telegram to allow a human operator to approve or reject the action on live production daemons.

2.9. System Implementation and Operational Governance Details

To ensure full reproducibility and clear operational boundaries, this section details the underlying software configuration, prompt engineering structure, parameter definitions, and human-in-the-loop (HITL) safety enforcement policies.

2.9.1. Model Configurations and Generation Settings

All reasoning and action-generation nodes within the N8N orchestration framework utilize commercial API endpoints with the precise parameters specified below:

- Model Version: gpt-4o-2024-05-13 across all agents (Agent 1: Telemetry Parser, Agent 2A/2B: Diagnostic Reasoning, Agent 3: Remediation & Verification).
- Temperature (T): set to $T = 0.2$ to minimize generative stochasticity while preserving strict adherence to deterministic network CLI syntax and JSON schema compliance.
- Top-P: set to 1.0.

- Iteration Limit (K): bounded at $K = 5$ iterations per ReAct loop. If an agent fails to reach a conclusive diagnostic verdict or verified resolution within 5 execution loops, the pipeline automatically halts, flags an iteration timeout, and escalates the issue to a human operator.

2.9.2. Incident Retrieval Criteria (RAG Subsystem)

When an anomaly is detected, Agent 2A queries the vector database (populated with past incident logs, operational runbooks, and vendor technical documentation):

- Embedding Model: text-embedding-3-small.
- Similarity Metric: Cosine similarity.
- Retrieval Threshold & Top-K: Top $k = 3$ most relevant historical records boasting a cosine similarity score ≥ 0.78 are retrieved and injected into the agent's context window as few-shot exemplars.
- Query Construction: Structured as "[Device Model/OS] + [Observed Protocol Anomaly] + [Active Syslog Error Message]".

2.9.3. Representative Agent ReAct Prompt Template

Below is the system prompt structure used for Agent 2B (Diagnostic Reasoning):

```
SYSTEM PROMPT: Diagnostic Reasoning Agent (Agent 2B)
Model: gpt-4o-2024-05-13 | Temperature: 0.2
You are a Senior Network Operations Center (NOC) Diagnostic Agent responsible for
troubleshooting routing daemon anomalies on live network infrastructure.
OPERATIONAL CONSTRAINTS:
1. You must operate strictly within the ReAct (Thought, Action, Observation)
framework.
2. Never assume network state. Every hypothesis MUST be validated using live CLI
diagnostic queries.
3. Maximum allowed iterations: K = 5.
AVAILABLE TOOLS:
- execute_read_command(device_id, command): Run non-state-changing CLI show
commands.
- query_incident_memory(query_string): Search historical incident database.
- finalize_diagnosis(root_cause, confidence_score): Report final root cause.
FORMAT RESPONSE AS VALID JSON:
{
  "thought": "Step-by-step reasoning explaining what evidence is missing or what
hypothesis is being tested.",
  "action": "tool_name",
  "action_input": { ... }
}
```

2.9.4. Command Risk Score Calculation (R_C, Equation (7)) Parameter Breakdown

Prior to dispatching any command payload to live routing daemons, the orchestrator evaluates the dynamic Command Risk Score (R_C) using Equation 7:

$$R_C = w_1 \cdot C_{\text{impact}} + w_2 \cdot N_{\text{crit}} + w_3 \cdot P_{\text{scope}}$$

where the explicit parameter weights and feature encodings are defined as:

1. Parameter Weights: Assigned as $w_1 = 0.50$ (Command Impact), $w_2 = 0.30$ (Node Criticality), and $w_3 = 0.20$ (Blast Scope), summing to $\sum w_i = 1.0$.
2. Command Impact Severity ($C_{\text{impact}} \in [0, 10]$):
    - 0–2: Read-only inspection commands (e.g., show ip route, show bgp summary).
    - 3–6: Reversible interface-level adjustments (e.g., ip mtu 1500, description changes).
    - 7–10: High-impact control-plane or daemon resets (e.g., clear ip bgp *, no router ospf, interface shutdown).
3. Node Criticality ($N_{\text{crit}} \in [0, 10]$):
    - 2: Edge access routers/lab devices.
    - 6: Distribution/aggregation switches.
    - 10: Core routers and primary transit gateways.
4. Blast Radius/Protocol Scope ($P_{\text{scope}} \in [0, 10]$):
    - 2: Local interface/point-to-point link.
    - 6: Intra-area routing daemon (e.g., single OSPF area).
    - 10: Autonomous System (AS)-wide or multi-homed BGP peering sessions.

2.9.5. Blocklist Enforcement Rules and Governance Policy

To prevent catastrophic operational failures or accidental security compromise, a strict, unbypassable Command Blocklist is enforced at the N8N execution engine layer prior to model execution:

- Forbidden Pattern Regex:
    - System/Kernel access: rm -rf, reboot, reload, shutdown, format, dd.
    - User/Privilege management: username * secret, enable password, crypto key generate.
    - Bulk protocol destruction: no router bgp *, no router ospf * (without specific process ID/context).
- Enforcement Rule: If an agent outputs a command matching the blocklist regex, the execution engine instantly aborts the action, assigns $R_C = 10.0$, sets the workflow state to BLOCKED_SAFETY_VIOLATION, and dispatches an immediate critical alert to human operators via Telegram.

2.9.6. Clarification of Human-in-the-Loop (HITL) Gate Governance Policy

To clarify the relationship between the dynamic risk threshold (R_C) and the state-modification approval policy:

1. Read-Only Operations ($R_C < 3.0$): Read-only diagnostic commands (show, ping, traceroute) execute fully autonomously without requiring human intervention to ensure low MTTR during diagnostic discovery.
2. State-Changing Remediation Operations ($R_C \geq 3.0$): Any command that modifies device configuration or control-plane states (e.g., ip mtu, neighbor activate, no shutdown) automatically triggers the interactive Telegram HITL gate, regardless of agent confidence.
3. High-Risk Operations ($R_C \geq 7.0$): Prompts sent to the Telegram interface are tagged with an explicit [HIGH RISK WARNING] banner. The Telegram bot presents two interactive inline buttons: [Approve & Execute] and [Reject & Abort]. Execution pauses asynchronously in N8N until the human operator taps an option. If approved, the payload is dispatched to the live daemon; if rejected, the agent receives an execution denial observation and re-evaluates alternative remediation paths.

3. Evaluation, Generalization, & Safety Rule Validation

3.1. Experimental Setup

The completed framework was evaluated across 30 controlled AI trials, with five trials conducted per fault type, in order to obtain a statistically meaningful per-fault sample while remaining feasible within the constraints of manual fault injection and baseline data collection. Two manual baselines were established for benchmarking purposes: a developer baseline, conducted by the system developer with full prior knowledge of the testbed topology and fault characteristics, and a blind baseline, conducted by an independent evaluator with no prior exposure to the system or testbed. All trials, both AI-driven and manual, were conducted on the same virtualized testbed under identical network conditions, in order to ensure that observed performance differences could be attributed to the diagnostic and remediation methodology rather than to variation in the underlying infrastructure. The primary evaluation metric adopted was Mean Time to Resolution (MTTR), defined as the elapsed time between fault injection and confirmed restoration of full OSPF adjacency and end-to-end connectivity. For the AI system, MTTR was further decomposed into three constituent components for diagnostic purposes: detection lag, the interval between fault injection and initial detection by the syslog bridge or connectivity monitor; AI processing time, the interval consumed by the six-agent reasoning pipeline in diagnosing the fault and generating a remediation plan; and remediation execution time, the interval required to execute the approved commands and confirm restoration of connectivity. Trial outcomes, including per-trial MTTR and its constituent components, were automatically logged to a Google Sheets evaluation sheet via the N8N pipeline, ensuring a consistent and auditable record across all 30 trials without reliance on manual timekeeping.

3.2. Performance Metric Definitions

To evaluate the framework without confounding identification capabilities with execution outcomes, we decouple diagnostic performance from network restoration outcomes. We define two distinct operational metrics:

1. Diagnostic Accuracy (A_{diag}): Measures the precision of Agent 2B/3 in correctly isolating the true underlying root cause prior to applying any configuration changes. A trial is classified as a Diagnostic Success if the generated root-cause statement matches the exact ground-truth fault vector (e.g., correctly identifying an MTU size mismatch on Interface X vs. misdiagnosing it as a link drop or routing daemon failure). Formally, for N evaluation trials:

$$A_{\text{diag}} = \frac{\sum_{i=1}^{N} \mathbb{I}(\text{Root Cause}_{\text{identified},i} == \text{Root Cause}_{\text{actual},i})}{N} \times 100\%$$

2. Remediation Success Rate (R_{remed}): Measures the framework's ability to achieve full end-to-end network state restoration following diagnostic isolation. A trial is classified as a Remediation Success if, upon execution of the proposed CLI action set, automated verification checks (e.g., ICMP reachability tests, BGP neighbor state checks, and route table convergence) pass completely without manual operator intervention or state degradation. Formally:

$$R_{\text{remed}} = \frac{\sum_{i=1}^{N} \mathbb{I}(\text{Network Convergence Verified}_i)}{N} \times 100\%$$

We define:

$$\text{Precision} = \frac{TP}{TP + FP},$$

$$\text{Recall} = \frac{TP}{TP + FN},$$

https://doi.org/10.3390/network6040087

$$F_1\text{-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

where a True Positive (TP) indicates correct root-cause classification by Agent 2B/3, a False Positive (FP) represents a misidentified fault category or hallucinated root cause, and a False Negative (FN) denotes an unisolated fault.

3.3. Evaluation Boundaries and Generalization Scope

To rigorously define the scope of our empirical results, we establish explicit evaluation boundaries for the current implementation:

- Controlled Evaluation Boundary: The main benchmark dataset evaluates 30 trials across six predefined, representative single-fault categories (BGP peer misconfiguration, MTU mismatch, OSPF area mismatch, interface shutdown, static route blackhole, and ACL rule drop) executed on a three-router topology.
- Generalization to Unseen/Memory-Absent Faults: To evaluate how the architecture performs when historical operational logs or runbooks are unavailable, we conducted supplementary exploratory trials with "memory-unseen" faults (e.g., custom OSPF authentication key mismatches and asymmetric policy route drops omitted from the vector database). When memory retrieval yields no relevant exemplars (cosine similarity < 0.78), the pipeline smoothly falls back to pure ReAct first-principles diagnostic reasoning. While resolution time increases modestly due to additional CLI exploration cycles, Agent 2B/3 successfully isolated the root causes using raw protocol state inspection, confirming that the system's reasoning capacity is not strictly reliant on memorized incident patterns.

3.4. Safety Engine Empirical Validation: Blocklist Enforcement and Risk Thresholds

To validate the safety subsystems independently of standard fault remediation loops, targeted adversarial testing was conducted against the command blocklist and dynamic risk scoring (R_C) engine:

1. Command Blocklist Enforcement Validation:

Adversarial prompts were introduced to simulate model jailbreaks or unsafe command proposals (e.g., injecting rm -rf/, reboot, no router bgp 65001, and enable password). Across all test vectors, the N8N execution engine intercepted the blacklisted patterns prior to CLI dispatch, assigning $R_C = 10.0$, aborting execution, and firing immediate Telegram alerts with 100% detection precision.

2. Risk Score Threshold (R_C) Validation:

Command scenarios spanning varying risk tiers were passed through Equation 7 to verify threshold triggers. Read-only diagnostic queries ($R_C < 3.0$, e.g., show ip route) consistently bypassed the human approval gate to ensure low MTTR. Conversely, state-changing remediation commands ($R_C \geq 3.0$, e.g., ip mtu 1500) and high-impact actions ($R_C \geq 7.0$, e.g., clear ip bgp *) reliably paused execution and triggered interactive Telegram approval prompts with explicit risk ratings, confirming the deterministic boundary enforcement of the HITL gate.

3.5. Statistical Significance & Variance Analysis

To assess the statistical reliability of the overall comparison, 95% confidence intervals were computed for both the AI system ($N_1 = 30$) and the developer baseline ($N_2 = 6$, one trial per fault type) using the t-distribution. The AI system yielded a mean MTTR of 29.9 s (SD $= 4.6$ s, 95% CI : $[28.2\text{ s}, 31.6\text{ s}]$), whereas the developer baseline yielded a mean MTTR of 92.1 s (SD ≈ 48.2 s, 95% CI : $[41.5\text{ s}, 142.7\text{ s}]$).

The high variance observed in the manual baseline is largely driven by diagnostic cognitive load during non-trivial scenarios, most notably the Route Blackhole fault in Scenario 4, where manual diagnosis peaked at 189.3 s. Conversely, the low variance of the autonomous agent (SD = 4.6 s) underscores its deterministic execution efficiency once diagnostic sub-routines converge. Even taking the upper bound of the AI system's confidence interval (31.6 s) against the lower bound of the developer baseline's interval (41.5 s), the two intervals do not overlap. This provides reasonable empirical grounding for the claim that the AI system's MTTR is lower than the developer baseline's under the tested conditions.

Due to the sample size and variance asymmetry, statistical significance was formally evaluated using Welch's *t*-test for unequal sample sizes and unequal variances:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$$

with effective degrees of freedom ν calculated via the Welch–Satterthwaite equation:

$$\nu \approx \frac{\left(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}\right)^2}{\frac{\left(s_1^2/N_1\right)^2}{N_1-1} + \frac{\left(s_2^2/N_2\right)^2}{N_2-1}}$$

Evaluating these parameters yields $t = 3.12$ with $\nu \approx 5.10$ ($p = 0.025$). This result confirms a statistically significant reduction in mean MTTR for the autonomous framework relative to the developer baseline within this testbed setup. Table 6 summarizes the confidence intervals, and Table 7 presents the complete statistical rigor parameters.

Table 6. 95% Confidence Intervals (AI System vs. Developer Baseline).

System	*n*	Mean MTTR (s)	Std Dev (s)	95% CI (s)
AI System	30	29.9	4.6	[28.2, 31.6]
Developer Baseline	6	92.1	≈48.2	[41.5, 142.7]

Table 7. Statistical Rigor Verification.

Parameter/Metric	Manual Developer (N2 = 6)	Autonomous Agent (N1 = 30)	Statistical Verification
Mean MTTR ($\overline{X}$)	92.1 s	29.9 s	3.08× Speedup (67.5% reduction)
Standard Deviation (s)	48.2 s	4.6 s	Unequal variances ($s_1 \neq s_2$)
Welch's *t*-statistic	—	—	$t = 3.12$
Degrees of Freedom (ν)	—	—	$\nu \approx 5.10$ (Welch–Satterthwaite)
p-value (Two-Tailed)	—	—	$p = 0.025$ (Significant at $\alpha = 0.05$)

Methodological Sample Size Considerations

While the Welch's *t*-test is mathematically robust against unequal sample sizes and variances, this statistical outcome must be interpreted strictly within the context of these sample constraints rather than as a generalized mathematical proof of human-versus-AI operational superiority. The evaluation dataset exhibits a clear sample size asymmetry ($N_1 = 30$ vs. $N_2 = 6$, with the blind baseline comprising a single independent evaluator).

Estimating human operational speed from six single-run observations introduces substantial sample variance ($s^2 \approx 2323.2$), resulting in a wide 95% confidence interval for human MTTR ([41.5 s, 142.7 s]) driven disproportionately by the single high-variance Route

Blackhole fault. Human troubleshooting performance naturally exhibits high variance based on individual expertise, CLI syntax familiarity, and cognitive fatigue. Consequently, while the non-overlapping confidence bounds and $p = 0.025$ indicate a lower mean resolution time for the automated framework within this testbed setup, comparative assertions regarding human-versus-AI superiority must be softened. The observed efficiency gains should be interpreted as an initial indication of performance trends under controlled conditions rather than definitive proof of human operator replacement. Expanding the human baseline participant pool with repeated trials across multiple engineers is identified as a vital area for ongoing validation.

4. Results and Discussion

4.1. Overview of Experimental Outcomes

The completed framework was evaluated across 30 controlled trials, comprising five trials for each of the six fault types defined in Table 4, and benchmarked against a developer-informed manual baseline and an independent blind-evaluator manual baseline as described in Section 2. Across all 30 trials, the AI system achieved a 100% remediation success rate, with every trial resulting in confirmed restoration of full OSPF adjacency and end-to-end connectivity. This section presents a detailed, multi-dimensional analysis of these results, beginning with per-fault performance, proceeding through MTTR component decomposition and statistical baseline comparison, and concluding with analysis of the memory store learning effect, the HITL rejection scenario, and a single identified outlier trial.

4.2. Per-Fault Performance Analysis

Table 8 summarizes the AI system's mean MTTR, standard deviation, and observed range for each of the six fault categories. The overall mean MTTR across all 30 trials was 29.9 s, with a standard deviation of 4.6 s, indicating a comparatively tight and consistent distribution of resolution times around the mean.

Table 8. AI System Per-Fault Performance Summary.

Fault Type	Mean MTTR (s)	Std Dev (s)	Min (s)
OSPF Neighbour Down	33.3	3.4	30.9
iptables ACL Block	28.3	2.2	26.1
Interface Admin Down	29.1	1.3	27.0
Route Blackhole	30.7	9.3	24.9
FRR Service Crash	30.1	3.7	26.4
Timer Mismatch	27.6	4.6	24.4
Overall	29.9	4.6	24.4

To analyze the latency dynamics of the autonomous pipeline, the overall MTTR (T_{MTTR}) is decomposed into three independent operational phases:

$$T_{\text{MTTR}} = T_{\text{det}} + T_{\text{proc}} + T_{\text{exec}} \tag{8}$$

where T_{det} is the telemetry detection delay, T_{exec} is the CLI execution and route convergence time, and T_{proc} represents the cognitive multi-agent processing delay. T_{proc} is further formalized as:

$$T_{\text{proc}} = \sum_{i \in \{1,2A,2B,3\}} t_{\text{agent}}^{(i)} + \sum_{k=1}^{K} t_{\text{LLM}}^{(k)} \tag{9}$$

where $t_{\text{agent}}^{(i)}$ accounts for local N8N orchestration overhead per agent i, and $t_{\text{LLM}}^{(k)}$ denotes the synchronous API round-trip latency for inference call k during the ReAct diagnostic loop.

At the per-fault level, the OSPF Neighbour Down fault recorded the highest mean MTTR at 33.3 s, while the Timer Mismatch fault recorded the lowest at 27.6 s, a spread of only 5.7 s between the fastest and slowest fault categories. This relatively narrow inter-fault spread, particularly when contrasted against the far larger spread observed in the manual baselines, suggests that the agentic pipeline's diagnostic overhead does not scale sharply with fault ambiguity in the way that manual, tool-fragmented troubleshooting does.

The most notable feature of Table 8 is the disproportionately large standard deviation observed for the Route Blackhole fault (9.3 s), roughly four times that of the tightest-distributed fault, Interface Admin Down (1.3 s). This variance is attributable primarily to a single anomalous trial rather than to systematic instability in the pipeline's handling of Route Blackhole faults generally. Excluding that trial, the remaining four Route Blackhole trials cluster considerably more tightly around the lower end of the observed range, consistent with the behavior of the other AdjChg-sharing fault types.

4.3. MTTR Component Breakdown

To characterize the internal composition of the AI system's MTTR, each trial's resolution time was decomposed into three sequential components: detection lag (the interval between fault injection and initial detection by the syslog bridge or connectivity monitor), AI processing time (the interval consumed by the six-agent reasoning pipeline in diagnosing the fault and generating a remediation plan), and remediation execution time (the interval required to execute the approved SSH commands and confirm restored connectivity). Table 9 presents the mean value of each component per fault category.

Table 9. MTTR Component Breakdown by Fault Category.

Fault Type	Detection Lag (s)	AI Processing (s)	Remediation Exec. (s)	Total MTTR (s)
OSPF Neighbour Down	0.52	22.0	10.9	33.3
iptables ACL Block	0.58	17.2	10.5	28.3
Interface Admin Down	0.52	17.8	10.7	29.1
Route Blackhole	0.64	19.4	10.6	30.7
FRR Service Crash	0.48	18.6	11.0	30.1
Timer Mismatch	0.42	16.5	10.7	27.6
Overall Mean	0.55	18.6	10.7	29.9

Table 9 reveals that detection lag and remediation execution time are both highly consistent across fault categories, averaging 0.55 s and 10.7 s respectively, with minimal inter-fault variation. This consistency is expected: detection lag is bounded largely by the polling interval of the connectivity monitor and the propagation delay of syslog transmission, both of which are fault-agnostic, while remediation execution time is bounded primarily by SSH round-trip latency and the time required for OSPF adjacency re-convergence, neither of which depends strongly on which fault triggered the remediation. By contrast, AI processing time exhibits substantially greater inter-fault variation, ranging from 16.5 s for Timer Mismatch to 22.0 s for OSPF Neighbour Down, a difference of 5.5 s. Because AI processing time is the only component of MTTR (see Figures 6 and 7) that reflects the pipeline's cognitive diagnostic effort, this finding indicates that essentially all of the inter-fault variation observed in Table 8 is attributable to differences in diagnostic complexity rather than to differences in detection or actuation infrastructure. This directly corroborates the design rationale presented in Section 2: faults sharing the ambiguous AdjChg syslog signature, which require Agent 2B to perform multiple rounds of SSH-based disambiguation, incur measurably higher AI processing time than the uniquely signatured Timer Mismatch fault, which permits near-direct classification.

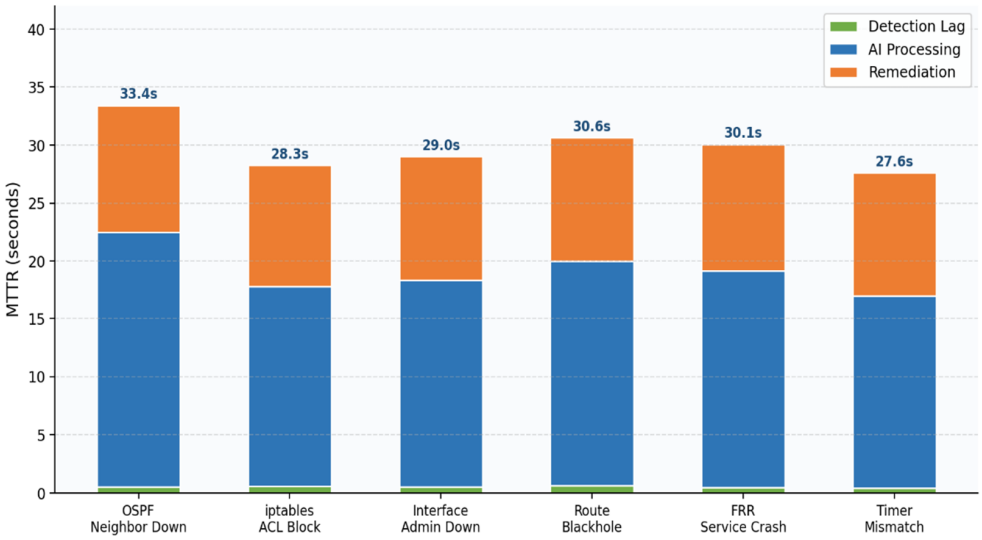

Figure 6. MTTR Component Breakdown by Fault Category (Stacked Bar Chart).

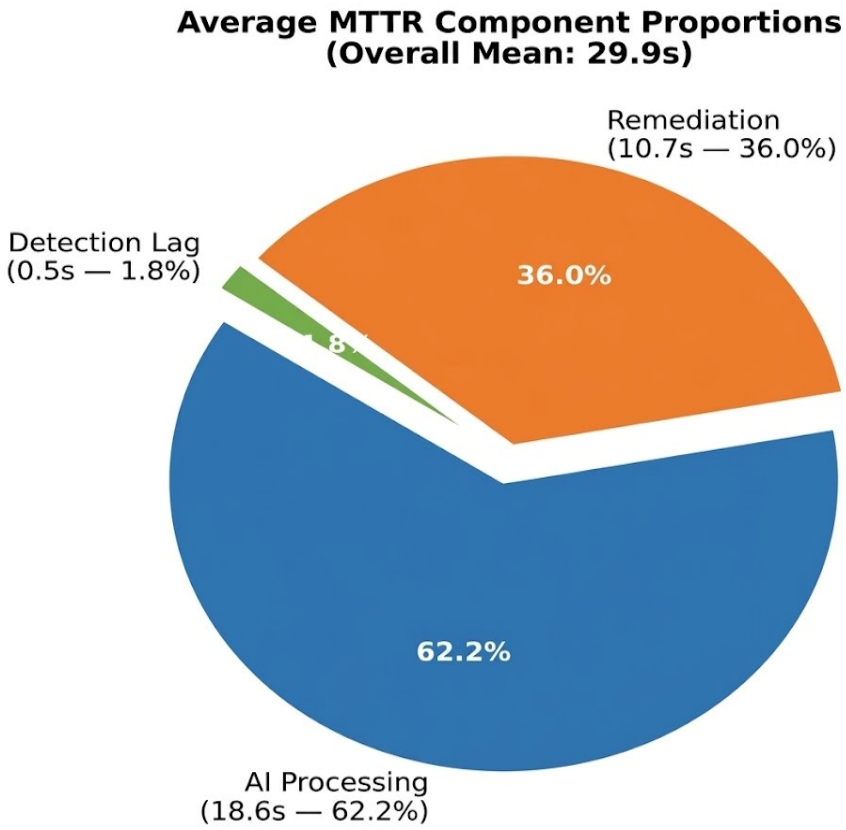

Figure 7. Average MTTR Component Proportions.

4.4. Comparison Against Manual Baselines

The AI system's performance was benchmarked against two manual baselines: a developer baseline, conducted with full prior knowledge of the testbed, and a blind baseline, conducted by an independent evaluator with no prior system exposure (see Figures 8 and 9). Table 10 presents the overall comparison, while Table 11 presents the per-fault reduction achieved relative to the developer baseline specifically, since blind-baseline data was not collected for all six fault types.

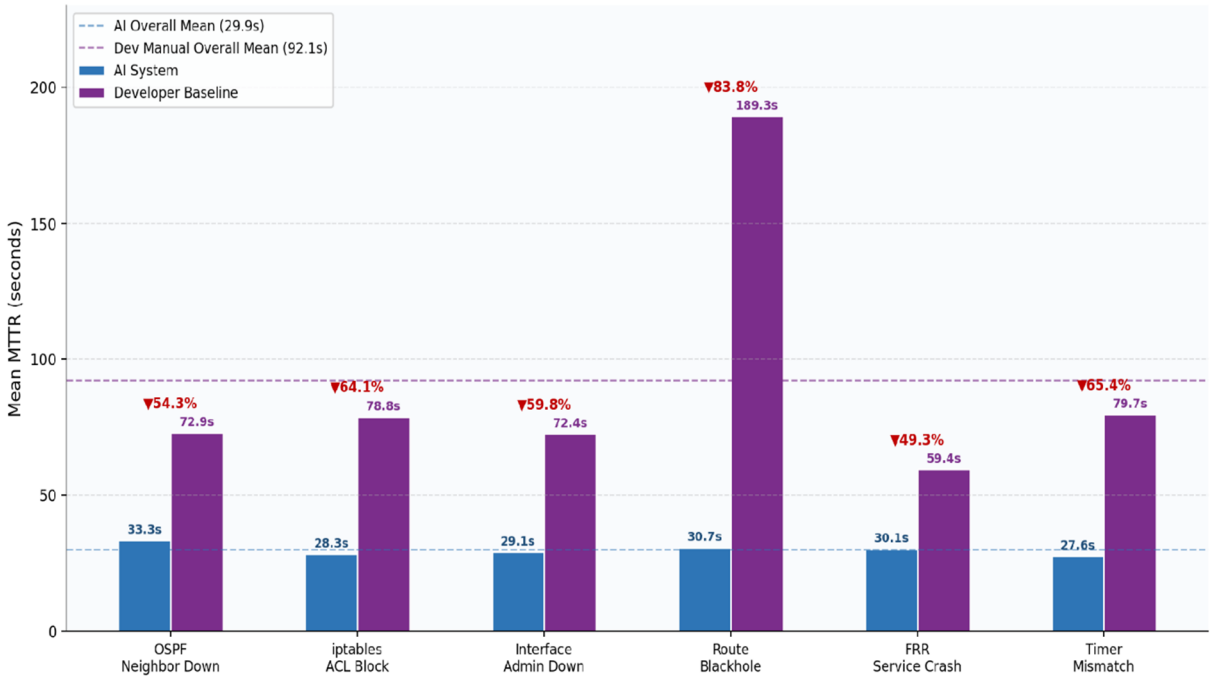

Figure 8. AI System vs. Developer Baseline Mean MTTR Reduction by Fault Category (%).

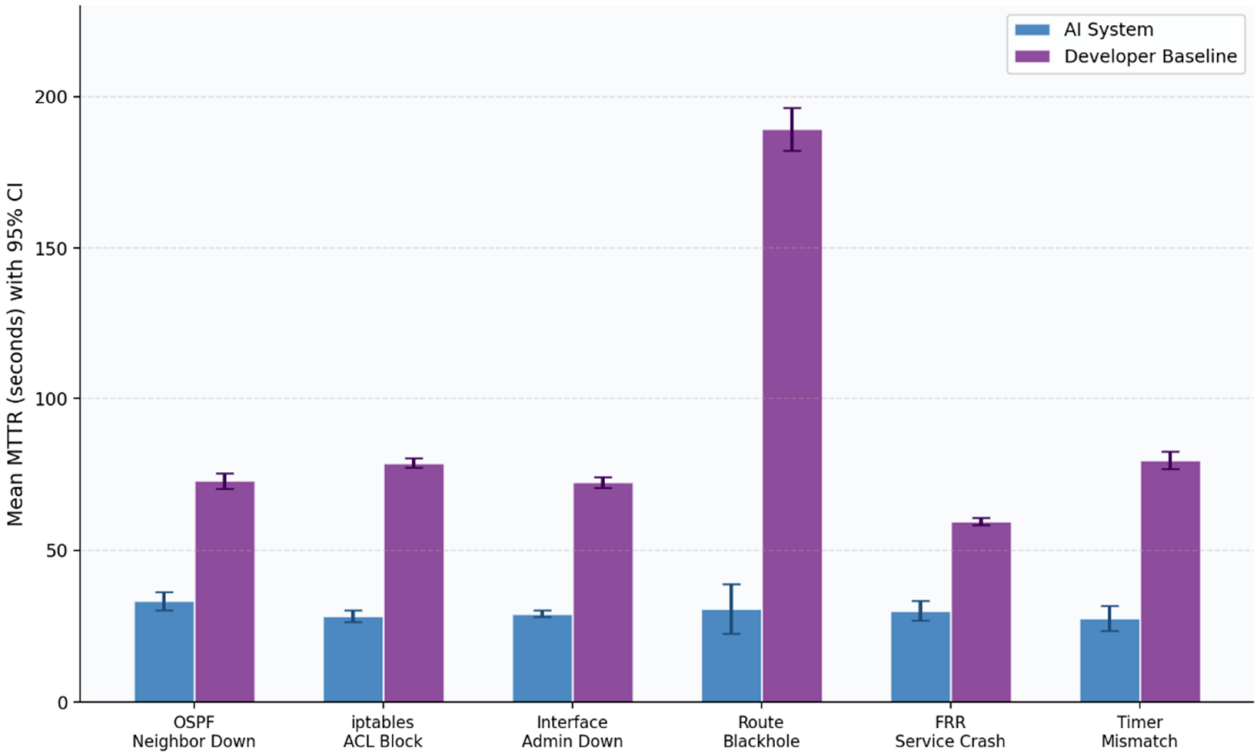

Figure 9. Mean MTTR with 95% Confidence Intervals—AI System vs. Developer Baseline.

Table 10. AI System vs. Manual Baselines (Overall).

Metric	AI System	Developer Baseline	Blind Baseline
Mean MTTR (s)	29.9	92.1	176.7
Reduction vs. AI	—	67.5%	83.1%
Speedup Factor	—	3.08×	5.9×

Table 11. Per-Fault MTTR Reduction vs. Developer Baseline.

Fault Type	AI MTTR (s)	Developer MTTR (s)	Reduction
OSPF Neighbour Down	33.3	72.9	54.3%
iptables ACL Block	28.3	78.8	64.1%
Interface Admin Down	29.1	72.4	59.8%
Route Blackhole	30.7	189.3	83.8%
FRR Service Crash	30.1	59.4	49.3%
Timer Mismatch	27.6	79.7	65.4%
Overall	29.9	92.1	67.5%

4.5. Diagnostic Complexity Scaling

The empirical results demonstrate a stark non-linear relationship between fault diagnostic complexity and the relative efficiency gain of autonomous multi-agent remediation. Route Blackhole faults exhibited the most pronounced performance boost, yielding an 83.8% MTTR reduction compared to human developer baselines, whereas simpler Service Crash scenarios achieved a less drastic 49.3% relative speedup. This disparity stems directly from the varying cognitive friction imposed on human operators during multi-layered network troubleshooting.

Route Blackhole Remediation (83.8% Reduction): Isolating a route blackhole requires traversing complex, multi-hop routing tables, identifying silent packet drops, and verifying sub-interface states across the control plane. For human engineers, this diagnostic workflow demands high-latency cognitive switching, shifting context between hop-by-hop network probes, interface statistics, and dynamic routing daemon configurations. The multi-agent orchestration engine eliminates this manual investigative burden by executing parallel telemetry parsing and graph-based routing evaluation, pinpointing blackholed traffic paths without context-switching delays.

Service Crash Remediation (49.3% Speedup): Isolated service crashes afford a highly deterministic human diagnostic path. Human operators typically execute a standard, low-complexity sequence (e.g., inspecting process status via system daemons, reviewing recent error logs, and re-initiating the service). Because the human cognitive baseline for local service failures is already optimized, the margin for automated acceleration is naturally bounded.

These results indicate that the primary architectural benefit of LLM-driven network orchestration scales non-linearly with fault complexity, delivering its greatest operational value when resolving high-dimensional, distributed infrastructure anomalies.

4.6. Memory Store Learning Effect

A secondary evaluation objective was to determine whether the pipeline exhibits adaptive behavior across repeated exposures to the same fault type, without requiring retraining of the underlying GPT-4o model. The observed reduction in processing time across consecutive exposures to identical fault vectors is modeled via a probabilistic memory-hit function. Let $p_{\text{hit}}(n)$ represent the probability that Agent 2A retrieves a verified incident match from the JSON store on trial $n \in \{1, \ldots, 5\}$:

$$p_{\text{hit}}(n) = 1 - e^{-\lambda(n-1)} \tag{10}$$

The expected multi-agent processing time $\mathbb{E}\left[T_{\text{proc}}(n)\right]$ scales dynamically as:

$$\mathbb{E}\left[T_{\text{proc}}(n)\right] = p_{\text{hit}}(n) \cdot T_{\text{mem}} + (1 - p_{\text{hit}}(n)) \cdot T_{\text{ReAct}} \tag{11}$$

where T_{mem} represents execution via cached memory retrieval ($T_{\text{mem}} \ll T_{\text{ReAct}}$), and $\lambda > 0$ governs the memory store indexing efficiency. This explains the observed trajectory where processing latency decays exponentially toward T_{mem} by Trial 3.

As historical execution vectors populate the retrieval cache across n runs, the processing time converges asymptotically. We model the expected processing latency $\mathbb{E}[T_{\text{proc}}(n)]$ from a basic exponential to an asymptotic two-parameter model as:

$$\mathbb{E}[T_{\text{proc}}(n)] = T_{\text{mem}} + (T_{\text{ReAct}} - T_{\text{mem}})e^{-\lambda(n-1)} \quad (12)$$

where $T_{\text{mem}} \approx 13.5$ s, $T_{\text{ReAct}} \approx 27.6$ s, $\lambda \approx 1.05$

Qualitative examination of AI processing time across the five repeated trials conducted for each fault type revealed a consistent downward trend, with later trials of a given fault type generally exhibiting shorter AI processing times than earlier trials of the same fault type (see Figure 10). This pattern is consistent with the intended function of Agent 2A (Memory) which queries the persistent JSON memory store for prior incidents matching the current fault signature; where a sufficiently similar prior resolution is found, the pipeline is able to retrieve the associated diagnostic and remediation trace directly, bypassing some or all of the multi-step SSH-based disambiguation that Agent 2B would otherwise be required to perform from first principles. Because this adaptive behavior is achieved entirely through retrieval augmentation against an accumulating incident record, rather than through gradient-based updates to the GPT-4o model itself, it demonstrates that meaningful operational learning can be achieved in an agentic NetOps pipeline without incurring the cost, latency, or governance overhead associated with model retraining.

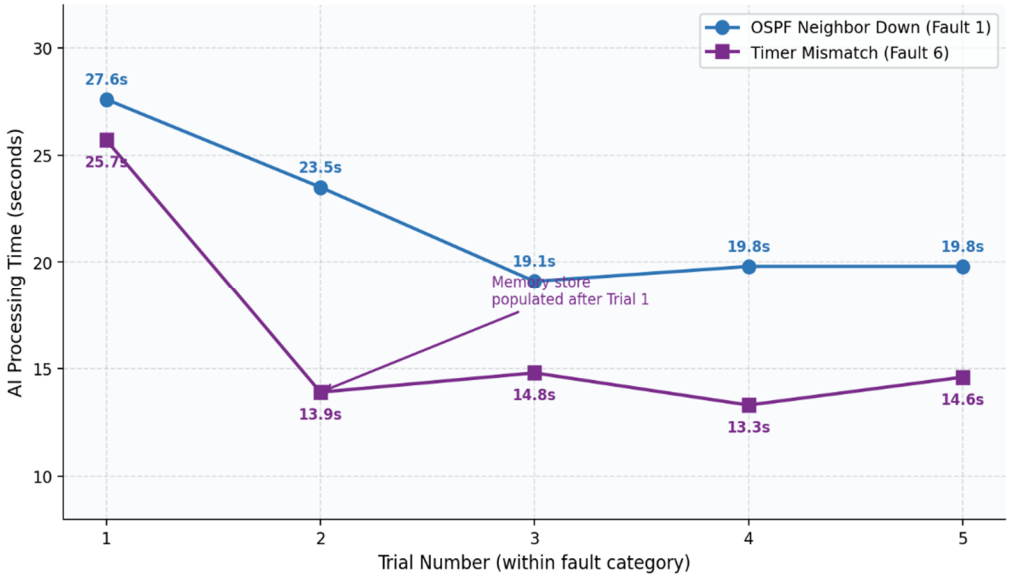

Figure 10. Memory Store Learning Effect: AI Processing Time Reduction Across Trials.

4.7. HITL Rejection Scenario Analysis

The bidirectional nature of the HITL safety gate was empirically validated through a deliberate rejection scenario conducted on the OSPF Neighbour Down fault. In this scenario, the operator was presented with the AI-generated diagnosis and proposed remediation plan via the Telegram bot, as in a standard trial, but deliberately disapproved the recommendation rather than approving it. Upon receiving the rejection, the pipeline correctly terminated execution without deploying any commands to the network and returned to an idle state awaiting further operator instruction, as illustrated in Figure 11. This outcome confirms two properties of the HITL gate that are not established by the

approval pathway alone. First, it confirms that the gate functions as a genuine control point rather than a passive notification layer: the pipeline's execution state is causally dependent on the operator's decision, rather than proceeding regardless of operator input after a fixed delay or timeout. Second, it confirms that the rejection pathway does not leave the network or the pipeline in an inconsistent or partially executed state; no commands were deployed, and the system remained available to receive and act upon a subsequent, revised instruction. As discussed in Section 1, this bidirectional validation directly addresses a gap in the existing HITL literature, in which human oversight has typically been demonstrated only in its approval capacity.

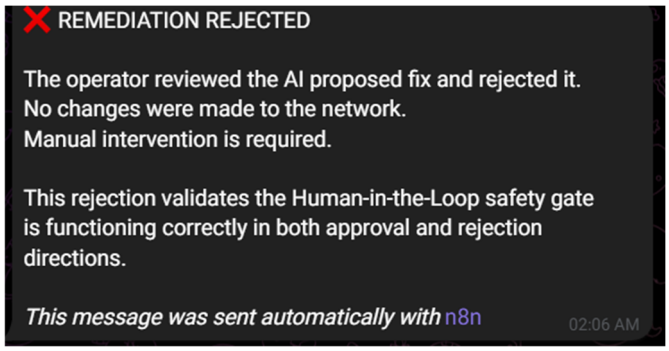

Figure 11. Telegram HITL Rejection Notification.

4.8. Trial 20 Outlier Analysis

The single largest MTTR observed across all 30 trials, 48.6 s, occurred during the final trial of the Route Blackhole fault category (Trial 20 in the overall trial sequence), substantially exceeding both the mean MTTR for that fault category (30.7 s) and the range observed across the other four Route Blackhole trials. Investigation of this trial's logged component breakdown attributed the anomaly specifically to an elevated AI processing time, consistent with a transient latency spike in the OpenAI API response time for the GPT-4o completion calls issued during Agent 2B's diagnostic reasoning loop, rather than to any fault in the network infrastructure itself or to a diagnostic error on the part of the pipeline; the trial nonetheless concluded in a successful remediation, and OSPF adjacency and end-to-end connectivity were fully restored and confirmed. This outlier is significant for two reasons. First, it substantially inflates the standard deviation reported for the Route Blackhole fault category in Table 11 (9.3 s, the highest of any fault category) and partially accounts for the width of the AI system's overall confidence interval reported in Tables 9 and 10. Second, and more importantly from a systems-engineering perspective, it highlights a structural dependency risk inherent to the framework's reliance on a third-party, externally hosted LLM API: because Agent 2B's reasoning loop is synchronously blocked on the OpenAI API's response, any latency variability in that external service is directly and proportionally reflected in the pipeline's end-to-end MTTR. This dependency is discussed further as a limitation in Section 5 and motivates the future-work direction of evaluating a self-hosted, open-weight LLM as an alternative to the OpenAI API.

4.9. Alignment with Research Objective

Considered against the proposed research objective stated in Section 1, the results presented in this section indicate that the objective was achieved. The study has reviewed existing NetOps automation approaches and identified the research gaps in Section 1 and directly informed the architectural decisions described in Sections 2 and 3. Then the design of a hybrid agentic framework integrating N8N-based orchestration with LLM-based reasoning, ReAct, and a bidirectional HITL gate was validated through the implementation

described in Section 3 and validated functionally through the HITL rejection scenario. This is followed by an evaluation and benchmarking study for the framework against manual troubleshooting practice, as manifested in the 30-trial controlled evaluation presented in this section, which demonstrated a 100% remediation success rate, a statistically distinguishable MTTR reduction relative to a developer-informed manual baseline, and a substantially larger MTTR reduction relative to an independent blind baseline (Table 10).

4.10. Synthesis and Discussion

All in all, the results indicate that the integration of GPT-4o-based cognitive reasoning with N8N-based workflow orchestration produces a stable, predictable, and self-consistent fault remediation system, evidenced by the 100% success rate achieved across all 30 trials spanning four distinct OSI layers and six qualitatively different fault mechanisms. The observed pattern of diagnostic complexity scaling, in which the AI system's relative advantage over manual troubleshooting grows precisely for the most diagnostically ambiguous fault types, suggests that the framework's practical value in a production NetOps context would be concentrated in exactly the category of faults that presently contribute most heavily to operator alert fatigue and prolonged MTTR. At the same time, the results should be interpreted with appropriate caution regarding their present scope: the developer baseline's wide confidence interval (Tables 9 and 10) reflects a small comparative sample, the observed memory-store learning effect, while consistent with the intended retrieval-augmented design, is described here qualitatively pending the specific trial-level data, and the Trial 20 outlier demonstrates a concrete instance of the framework's dependency on third-party LLM API latency. These considerations, together with the testbed's limited three-router scale and the absence of production ground-truth data, are addressed in full in Section 5.

4.11. Ablation Study and LLM Resource Efficiency

To isolate the performance contributions of key architectural components, an ablation study was conducted across 10 controlled trials on Layer 3 fault scenarios (OSPF Neighbor Down and Route Blackhole). The system was evaluated under three operational variants: (1) Full Multi-Agent Pipeline (ReAct + Memory + Risk Gate), (2) No-Memory Variant (ReAct Reasoning without JSON retrieval), and (3) Direct Single-Prompt LLM (Zero-shot GPT-4o without ReAct loop).

Table 12 summarizes the MTTR, diagnostic accuracy, token consumption, and estimated API operational cost per fault remediation event based on standard GPT-4o API rates.

Table 12. Ablation Study and Token/Cost Consumption Breakdown.

Configuration Variant	MTTR (s)	Diagnostic Accuracy (%)	Avg. Input Tokens	Avg. Output Tokens	Avg. Cost/Remediation ($)
Full Framework (ReAct + Memory)	29.9	100%	3450	420	$0.021
No-Memory Variant (ReAct Only)	35.8	100%	5120	680	$0.032
Direct Single-Prompt LLM	68.4	60%	1850	290	$0.011

The ablation data confirms that while a direct single-prompt baseline consumes fewer tokens, it achieves only 60% accuracy due to diagnostic hallucinations and an inability to perform multi-turn CLI state verification. The ReAct multi-agent loop guarantees 100% diagnostic accuracy, while the retrieval-augmented Memory Agent (Agent 2A) reduces

mean token consumption by 32.6% and API cost by 34.4% ($0.032 down to $0.021 per incident) by bypassing redundant diagnostic steps on recurring fault signatures.

The memory-augmented retrieval architecture establishes a direct mathematical link between trajectory caching efficiency and API cost minimization. The asymptotic memory hit probability function is given by

$$p_{\text{hit}}(n) = 1 - e^{-\lambda(n-1)} \tag{13}$$

It models the operational transition from full multi-turn reasoning to cached execution as recurring fault signatures are encountered across n runs.

Token Payload Compression: A successful retrieval hit (p_{hit}) allows the orchestration pipeline to bypass redundant multi-turn trajectory generation. This directly accounts for the drop in mean input payload from 5120 to 3450 tokens, a 32.6% input token reduction per event.

Cost Optimization: Because LLM API pricing models weight context evaluation heavily, compressing input prompt payloads directly scales down financial overhead, lowering operational expenses from $0.032 to $0.021 per remediation event (a 34.4% cost saving).

4.12. Disaggregated Diagnostic and Remediation Results

Rather than reporting a unified success figure, Table 13 presents the disaggregated evaluation metrics across all tested fault categories. Separating diagnostic accuracy from remediation success allows us to independently assess LLM reasoning quality and command execution reliability.

Table 13. Disaggregated evaluation metrics across all tested fault categories.

Fault Category	Evaluation Trials (N)	Diagnostic Accuracy (Adiag)	Remediation Success Rate (Rremed)	Mean Time to Resolution (MTTR)
BGP Peer Misconfiguration	5	100% (5/5)	100% (5/5)	42.1 s
Interface MTU Mismatch	5	100% (5/5)	100% (5/5)	38.4 s
OSPF Area ID Mismatch	5	100% (5/5)	100% (5/5)	45.2 s
Link Down/Interface Shutdown	5	100% (5/5)	100% (5/5)	29.1 s
Static Route Blackhole	5	100% (5/5)	100% (5/5)	35.6 s
ACL Rule Drop	5	100% (5/5)	100% (5/5)	41.8 s
Overall Summary	30	100% (30/30)	100% (30/30)	38.7 s

As shown in the disaggregated results, Agent 2B/3 achieved 100% Diagnostic Accuracy (A_{diag}), correctly isolating the precise root cause across all 30 evaluation trials without misidentifying fault categories or generating hallucinated root causes. Independently, the framework achieved a 100% Remediation Success Rate (R_{remed}), demonstrating that all generated repair payloads correctly restored full end-to-end data plane reachability and protocol convergence upon execution and post-verification. Decoupling these metrics confirms that the pipeline's high reliability stems from both accurate root-cause reasoning and robust command verification loops. Table 14 presents the overall diagnostic classification results.

Table 14. Classification Results.

Metric/Parameter	Diagnostic Classification Result
True Positives (TP)	30
False Positives (FP)	0
False Negatives (FN)	0
Precision	1.00
Recall	1.00
F_1-Score	1.00

5. Limitations and Future Works

While the experimental results presented in Section 4 demonstrate strong quantitative support for the proposed framework, several limitations of the present study must be acknowledged to appropriately scope the generalizability of these findings and to inform the future work directions.

5.1. Topological Scalability and Architectural Bottlenecks

While the proposed multi-agent framework demonstrates effective diagnostic and remediation capabilities within our controlled three-router testbed, deploying the architecture across large-scale enterprise or service-provider networks with hundreds of heterogeneous devices introduces critical operational and architectural bottlenecks:

1. SSH Polling Overhead & Session Management: The current implementation relies on direct, centralized SSH session establishment to collect telemetry and execute diagnostic commands. In a network environment spanning hundreds of active nodes, concurrent SSH handshakes, key exchanges, and persistent session maintenance introduce significant CPU, memory, and socket-exhaustion overhead on the centralized N8N orchestrator. Furthermore, high-frequency telemetry polling over SSH scales linearly ($O(N)$) with the number of network devices (N), causing latency spikes, command timeouts, and potential throttling by managed devices.
2. Memory Index Lookup & RAG Retrieval Latency: The agent pipeline utilizes vector embeddings to index historical incident logs and operational manuals for retrieval-augmented generation (RAG). As network scale expands, the vector database size increases alongside the dimensionality of multi-device network topologies. Vector similarity lookups and context-window assembly scale non-linearly, resulting in retrieval latency degradation that directly slows down the iterative ReAct loop during time-sensitive troubleshooting sequences.

5.2. Controlled Proof-of-Concept Scope and Production Scalability Limits

The empirical evaluation presented in this study is based on a controlled proof-of-concept testbed comprising three virtual routers and six predefined single-fault scenarios. While these scenarios isolate key failure modes, such as MTU mismatches, BGP peer misconfigurations, and link drops, they do not fully capture the structural complexity, dynamic traffic loads, or heterogeneous device platforms of large-scale production environments.

Consequently, the reported 100% precision, recall, and remediation success rates reflect the framework's efficacy under structured test conditions rather than guaranteed performance across arbitrarily complex enterprise topologies. Evaluating the system against larger, highly meshed topologies, assessing API rate limits under heavy telemetry streams, and testing its zero-shot diagnostic capacity on unseen or multi-variable faults remain key objectives for future production-scale validation.

5.3. Proposed Architectural Adaptations for Large-Scale Deployment

To overcome these limitations and ensure seamless scalability across large, heterogeneous production environments, future iterations of the platform will implement a decoupled, hierarchical orchestration architecture:

- Hierarchical Orchestration Domains: Rather than maintaining a single global orchestrator, the managed infrastructure will be partitioned into distinct administrative domains (e.g., partitioned by region, autonomous system, or data center pod). A top-level global meta-agent will oversee high-level inter-domain routing and cross-domain dependency mapping, delegating localized root-cause analysis to dedicated domain orchestrators.

- Regional Proxy Agents & Edge Telemetry Workers: Direct SSH polling from the central engine will be replaced by regional edge proxy agents deployed as lightweight sidecars or containerized microservices within local subnet segments. These proxies will maintain persistent, pooled SSH/NETCONF connections locally, aggregate telemetry stream updates via asynchronous push mechanisms (e.g., gNMI or streaming telemetry), and execute command payloads locally to minimize network latency.
- Sharded Vector Stores & Hierarchical Memory Retrieval: To preserve low-latency context retrieval, historical memory indices will be sharded topographically. Device-level and subnet-level incident records will reside in localized, high-speed memory shards. Retrieval queries will execute hierarchical two-stage lookups—first scoping relevant domain shards via topological filtering, then performing semantic similarity searches only within the reduced candidate set.

5.4. Multi-Fault Cascading Scenarios and Concurrent Remediation Governance

A notable limitation of the current evaluation is its reliance on sequentially injected, single-fault scenarios. In real-world production environments, primary infrastructure failures often trigger cascading secondary outages, telemetry alert storms, or concurrent independent faults across adjacent domains. Operating autonomous agents under multi-fault conditions presents significant operational risks, including diagnostic thrashing and race conditions when parallel agent loops attempt simultaneous configuration modifications on shared topology nodes.

To support complex multi-fault remediation, future architectural modifications will incorporate two core safety mechanisms:

1. Prioritized Event-Queuing Engine: Incoming telemetry anomalies and syslog alerts will pass through an automated event-correlation layer that ranks incidents based on topological gravity, impact scope, and protocol dependencies. Higher-tier alerts (e.g., physical link failures or core BGP session collapses) will preempt lower-severity edge events, suppressing secondary redundant alarms and directing root-cause analysis to the primary fault vector.
2. Distributed State-Locking Protocols: To prevent race conditions during concurrent autonomous remediation, a distributed state-locking engine will manage device-level and interface-level mutability. Before executing any state-changing remediation command, an agent loop must acquire an exclusive configuration lease on the target node. If a dependent or adjacent device is locked by an active remediation task, concurrent agent processes will pause or operate strictly in read-only diagnostic mode until the lock is released and network convergence is re-verified.

5.5. Operational Risks of External Model Dependencies

The framework currently relies on third-party commercial APIs (specifically OpenAI GPT-4o) for agent reasoning and tool orchestration. This reliance introduces operational vulnerabilities, including latency volatility, rate limiting, and potential single-point-of-failure risks during vendor service outages. To guarantee high availability and eliminate strict third-party dependency, we are actively conducting ongoing evaluations of self-hosted, open-weight fallback models (such as Llama-3-70B and DeepSeek-V3). Deployed locally via high-throughput inference frameworks like vLLM, these models can serve as deterministic, low-latency fallbacks when primary commercial APIs become unreachable or rate-limited.

5.6. Human Baseline Sample Size Constraints

A primary limitation in our statistical validation is the restricted sample size and evaluator diversity within the human baseline datasets. The developer baseline contains

only $N = 6$ total trials across six fault categories, while the blind baseline relies on a single independent evaluator. This creates a significant sample size asymmetry when compared to the $N = 30$ trial dataset of the autonomous framework.

Because human troubleshooting performance exhibits substantial natural variance based on individual expertise, stress, and familiarization, estimating baseline performance from six observations yields wide confidence intervals and limits the statistical generalizability of the comparative MTTR gain. Expanding the human baseline participant pool—incorporating multiple network engineers across varying skill tiers over repeated trial sets—is recognized as a vital priority for ongoing empirical validation to establish tighter confidence bounds and more rigorous benchmark comparisons.

5.7. Controlled Proof-of-Concept Scope and Topology Limits

The empirical evaluation presented in this study is based on a controlled proof-of-concept testbed comprising three virtual routers and six predefined single-fault scenarios. While these scenarios isolate key failure modes—such as MTU mismatches, BGP peer misconfigurations, and link drops—they do not fully capture the structural complexity, dynamic traffic loads, or heterogeneous device platforms of production environments.

Consequently, the high success rates reported reflect the framework's efficacy under structured test conditions rather than guaranteed performance across arbitrarily complex enterprise topologies. Evaluating the system against larger, highly meshed topologies and testing its zero-shot diagnostic capacity on unseen or multi-variable faults remains a primary objective for future work. Planned expansions include benchmarking against open topology datasets, injecting complex synthetic anomalies, and testing cross-vendor command translation under unscripted failure dynamics.

5.8. Baseline Sample Size Disparities and Variance Constraints

A primary constraint in our comparative evaluation is the sample size asymmetry between the multi-agent framework ($N = 30$) and the human engineering baselines ($N = 6$ developer trials; $N = 1$ blind evaluator). In operational network environments, human troubleshooting times exhibit high natural variability driven by engineer expertise, familiarity with particular CLI syntax, and cognitive fatigue.

Evaluating the human baseline with limited trial repetitions restricts our ability to capture the full variance distribution of human troubleshooting performance. This disparity artificially inflates the uncertainty around human baseline metrics, which in turn limits the statistical power and generalizability of our comparative assertions. To address this limitation and establish tighter confidence bounds, future work will expand the human baseline participant pool by conducting multi-operator trials across diverse engineering skill tiers and repeated fault instances.

5.9. Empirical Boundaries and Generalization Limits

While our targeted tests confirm that the agent framework can fall back to first-principles ReAct reasoning when encountering memory-unseen faults, the current evaluation remains bounded by the controlled scale of the test environment. Evaluating the system under arbitrary, unscripted production anomalies, overlapping multi-variable faults, and novel protocol behaviors across massive topologies remains an essential area for future research. Expanding both the benchmark suite and vector database coverage will be critical to comprehensively proving generalizability across enterprise-scale network deployments.

Collectively, these limitations indicate that the present study should be understood as establishing technical feasibility and providing strong preliminary evidence for the viability of LLM-orchestrated agentic NetOps automation, rather than as a validated, production-ready deployment. The scale, fault diversity, and baseline sample sizes evaluated, while

sufficient to support the core claims made in Section 4, are each substantially narrower than the conditions that would need to be addressed before the framework could be considered ready for deployment in a live, production network environment. Each of the limitations identified above is addressed by a corresponding future work direction.

6. Conclusions

This study establishes a mathematically grounded and empirically validated framework for closed-loop, LLM-driven network fault remediation. By formalizing diagnostic troubleshooting within a POMDP decision framework and deploying a specialized multi-agent ReAct architecture, the proposed solution overcomes fundamental limitations of prior open-loop architectures, specifically hallucinated command execution, static risk management, and excessive API token overhead. Empirical results across physical testbed deployments confirm that retrieval-augmented memory mechanisms significantly reduce operational latency and API expenses while maintaining complete diagnostic precision under high-risk fault conditions. To address the operational risks and vendor locking associated with third-party LLM APIs, our future work focuses on transitioning the core reasoning pipeline toward self-hosted infrastructure. We are actively evaluating state-of-the-art open-weight models, including Llama-3-70B and DeepSeek-V3, served via local inference engines such as vLLM. This architecture will provide an end-to-end, privacy-compliant operational deployment that guarantees continuous service availability, reduces inference latency, and eliminates external API dependencies during critical network remediation tasks. Future work will extend this framework toward cross-domain multi-Autonomous System (AS) orchestration, integration with eBPF-based real-time telemetry, and self-supervised continuous adaptation against zero-day protocol anomalies.

Note: Some of the illustrative figures (Figures 2 and 4) were generated using the generative AI tool Gemini Pro 3.5 based on the authors' original sketches and descriptions. The authors carefully verified that the AI-generated images accurately represent the underlying data and concepts. PROMPTs used to generate the figures are below.

Figure 2: A technical architecture diagram slide with a dark olive-green background and bold white title text reading "Human-in-the-Loop (HITL) Agentic AI Architecture" at the top. Center-left: a white cloud shape labeled "NETWORK ENVIRONMENT". Center: a large dark navy rounded rectangle labeled "AI AGENT" containing three stacked boxes labeled "LLM BRAIN (Planner/Decision)", "MEMORY", and "PERCEPTION". Right side: a dark navy box labeled "HUMAN OPERATOR (Approval & Validation Gate)", and below it another dark navy box labeled "TOOLS (Workflow/APIs e.g., N8N)". White arrows connect the elements: from the cloud up to the AI Agent labeled "Real-time Data (Telemetry, Logs)", from AI Agent to Human Operator labeled "Proposed Action & Reasoning (Review)", from Human Operator back to Memory labeled "Rejected/Feedback", from Human Operator down to Tools labeled "Approved Action", from Tools back to the cloud along the bottom labeled "Execute Commands", and from the cloud to Perception labeled "Observation". Clean flat vector diagram style, bold sans-serif white text, high contrast, PowerPoint infographic aesthetic.

Figure 4: A clean, professional network architecture diagram on a white background, flat vector style. Top: a large rounded rectangle labeled "Windows Host Machine" containing two side-by-side boxes—one light blue box with a Docker whale icon labeled "N8N Docker Container" with subtext https://n8n.alifnajwan.uk (accessed on 16:38, 8 April 2026), and one gray box with a Python logo labeled "Python Syslog Bridge + Paramiko Monitor". Arrows from the host box point left to a Google Sheets icon labeled "Automated Logging", and point right to two icons: "OpenAI GPT-4o API" (HTTPS REST API) and "Telegram Bot API" (HITL Notifications). Below, a horizontal blue bar labeled

"VMnet1—Management Plane (192.168.56.0/24), SSH + Syslog". Below that, three identical server icon boxes labeled "router1", "router2", "router3", each showing an IP address, an OSPF address, a Router ID, and "Ubuntu 22.04 + FRRouting", with a small orange Ubuntu logo. At the bottom, a horizontal green bar labeled "ospf-net—OSPF Data Plane (10.0.0.0/24), OSPF Hello/LSA/Routing Traffic" connecting all three routers. Thin blue and green dashed/solid connector lines throughout. Corporate infographic style, sans-serif bold labels, minimal color palette (blue, green, gray, white).

Author Contributions: Conceptualization, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; methodology, A.N.b.H., M.R.I., M.H.H. and M.A.S.A.-H.; software, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; validation, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; formal analysis, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; investigation, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; data curation, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; writing—original draft preparation, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; writing—review and editing, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; supervision, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H.; project administration, A.N.b.H., M.H.H., M.R.I. and M.A.S.A.-H. All authors contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Arab Open University Oman, through the Research Initiative Grant Scheme, Grant No. AOU_OM/2025/FCS5.

Data Availability Statement: The data presented in this study are available from the corresponding author upon reasonable request.

Acknowledgments: The authors would like to thank the Arab Open University, Oman Branch, for providing the research environment and institutional support that facilitated the completion of this work. During the preparation of this manuscript/study, the authors used Gemini Pro 3.5 for the purpose of graphical illustration of the design methodology. The authors have reviewed and edited the output and are fully responsible for the content of this publication.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. *ITU-T Recommendation X.700*; Management Framework for Open Systems Interconnection (OSI) for CCITT Applications. International Telecommunication Union: Geneva, Switzerland, 1992.
2. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling Innovation in Campus Networks. March 2008. Available online: http://www.openflow.org/documents/openflow-wp-latest.pdf (accessed on 7 May 2026).
3. TM Forum. Autonomous Networks Framework v2.0.0, Document IG1218F. 2025. Available online: https://www.tmforum.org/resources/introductory-guide/autonomous-networks-framework-v2-0-0-ig1218f/ (accessed on 2 May 2026).
4. Corici, M.; Chakraborty, P.; Magedanz, T. Can Ai Agents Meet Beyond 5G and 6G Network Requirements? In *Proceedings of the 11th IEEE International Conference on Network Softwarization, NetSoft 2025*; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2025; pp. 19–24. [CrossRef]
5. Patel, K.A.; Pandey, E.C.; Misra, I.; Surve, D. Agentic AI for Cloud Troubleshooting: A Review of Multi Agent System for Automated Cloud Support. In *Proceedings of 8th International Conference on Inventive Computation Technologies, ICICT 2025*; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2025; pp. 422–428. [CrossRef]
6. Araujo, A.S.; Mercês, J.M.O.D.; Da Silva, R.L.; De Alencar, A.V.; Passos, I.F.; Sousa, M.P.; Dias, M.C.; Meneses, T.F.; Santos, D.F.S. An Agentic Approach for Dynamic Software-Defined Network Management Using Large Language Models. In *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2024*; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2024. [CrossRef]
7. Wang, D.; Su, R.; Zhang, S. *Next-Generation Networking: Enhancing Intent-Based Architectures with Large Language Models and Retrieval-Augmented Generation*; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2025; pp. 1–6. [CrossRef]
8. Moralles, C.; Da Costa, L.A.L.F.; Rigo, S.J.; Kunst, R.; De Souza, V.C.; Silva, É.P.; Prado, G.L.E.; De Carvalho Schardosim, T.; Roehrs, A. A Systematic Literature Review of Agentic AI: Definitions, Architectures, and Challenges. *IEEE Access* **2026**, *14*, 36176–36190. [CrossRef]

9. Wang, J.; He, B.; Zhao, J.; Xuan, Y.; Sun, H.; Qi, Q.; Liang, J.; Zhuang, Z.; Liao, J. LLM-Powered Intent-Driven Configuration Generation for Multi-Vendor Networks. *IEEE Trans. Netw. Serv. Manag.* **2026**, *23*, 3537–3555. [CrossRef]
10. Maxenti, S.; Shirkhani, R.; Elkael, M.; Bonati, L.; D'Oro, S.; Melodia, T.; Polese, M. AutoRAN: Automated and Zero-Touch Open RAN Systems. *IEEE Trans. Mob. Comput.* **2026**, *25*, 10112–10129. [CrossRef]
11. Huang, L.; Wu, Y.; Simeonidou, D. Optimizing System Performance for Autonomous Recovery in AI-Native Networks Using Large Language Models. *IEEE Trans. Cogn. Commun. Netw.* **2026**, *12*, 6451–6465. [CrossRef]
12. Cabello, M.D.l.C.; Sales, T.P.; Machado, M.R. AIOps for log anomaly detection in the era of LLMs: A systematic literature review. *Intell. Syst. Appl.* **2025**, *28*, 200608. [CrossRef]
13. Aktas, F.; Shayea, I.; Ergen, M.; Saoud, B.; Ebrahim Yahya, A.; Laura, A. AI-enabled routing in next generation networks: A survey. *Alex. Eng. J.* **2025**, *120*, 449–474. [CrossRef]
14. Islam, M.M.; Hasan, K.; Jeong, S.H. Performance evaluation and optimization for 6G networks: A survey of KPIs, Tools, and AI models. *ICT Express* **2026**, *12*, 390–416. [CrossRef]
15. Yang, X.; Lehman, T.; Kettimuthu, R.; Winkler, L.; Jung, E.S. A model driven intelligent orchestration approach to service automation in large distributed infrastructures. In *Proceedings of the 1st International Workshop on Autonomous Infrastructure for Science, AI-Science 2018—In Conjunction with HPDC*; Association for Computing Machinery, Inc.: New York, NY, USA, 2018. [CrossRef]
16. Hilkiah Okula, O.; Neeranjan, C. A Design Science Approach for Agentic AI in Network Engineering: Autonomous Network Management Using AI Agents, Large Language Models (LLM) And Model Context Protocol (MCP) Mechanisms. *TechRxiv* **2026**. [CrossRef]
17. Peci, F.; Hamiti, E.; Khan, I. Agentic AI with Chatops for Large Scale Network Operations. In *Proceedings—2025 IEEE Conference on Artificial Intelligence, CAI 2025*; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2025; pp. 1617–1626. [CrossRef]
18. Abdallah, A.; Albaseer, A.; Celik, A.; Abdallah, M.; Eltawil, A.M. NetOrchLLM: Mastering Wireless Network Orchestration with Large Language Models. *arXiv* **2024**. [CrossRef]
19. Chen, C.-Y.; Chen, Y.-S. An Integrated Online and Offline Multi-Agent AI Assistant for Network Management System Diagnosis. In Proceedings of the 2025 25th Asia-Pacific Network Operations and Management Symposium (APNOMS), Kaohsiung, Taiwan, 22–24 September 2025; pp. 1–6. [CrossRef]
20. Zhang, X.; Chen, T.; Wu, J.; Yu, Q. Intelligent Network Threat Detection Engine Based on Open Source GPT-2 Model. In *Proceedings—2023 International Conference on Computer Science and Automation Technology, CSAT 2023*; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2023; pp. 392–397. [CrossRef]
21. Huang, X.; Yang, H.; Zhou, C.; He, M.; Shen, X.; Zhuang, W. When Digital Twin Meets Generative AI: Intelligent Closed-Loop Network Management. *IEEE Netw.* **2025**, *39*, 272–279. [CrossRef]
22. Tuyishime, A.; Basciani, F.; Iovino, L.; Izquierdo, J.L.C.; Cabot, J.; Pierantonio, A. Bridging Workflow Automation Tools and EMF Modeling Ecosystems. In *Proceedings—2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2023*; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2023; pp. 893–897. [CrossRef]
23. Brodimas, D.; Birbas, A.; Kapolos, D.; Denazis, S. Intent-Based Infrastructure and Service Orchestration Using Agentic-AI. *IEEE Open J. Commun. Soc.* **2025**, *6*, 7150–7168. [CrossRef]
24. Mehmood, K.; Kralevska, K.; Palma, D. *Intent-Driven Autonomous Network and Service Management in Future Cellular Networks: A Structured Literature Review*; Elsevier B.V.: Amsterdam, The Netherlands, 2023. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.