AI-Driven Trouble Ticket Generation and Misconfiguration Diagnosis in Enterprise Systems

Anonymous members

***Abstract*—Misconfigurations in enterprise IT systems remain a leading cause of service disruptions and vulnerabilities. In this paper, we present a self-adaptive AI agent that interactively collects diagnostic symptoms through natural language conversations and collaborates with an automated diagnosis system to identify root causes. The AI agent holds a dialog with the user to clarify the user complaint and then translates it into structured diagnostic tickets by mapping symptoms to relevant configuration variables (CVs) and assigning initial confidence scores. These tickets guide ConfExp, a confidence-driven diagnostic framework, which refines hypotheses through system-level test execution. The combined system significantly improves the diagnosis process. In evaluations across 75 fault scenarios in emulated enterprise environments, the system correctly identified misconfigured components in over 93% of cases while reducing the number of tests required by up to 47% compared to using ConfExp alone. This demonstrates the effectiveness of combining a large language model (LLM)-driven symptom elicitation with confidence-guided test execution for scalable, automated troubleshooting.**

***Index Terms*—AI-driven troubleshooting, Enterprise network misconfiguration, Root-cause analysis, Large language models.**

I. Introduction

Enterprise IT infrastructures have evolved into complex, multi-layered environments comprised of numerous interdependent applications, network devices, and cloud services [1]. While such complexity enables significant scalability and flexibility, it also creates ample opportunities for misconfiguration errors. Misconfigurations remain a prominent source of system disruptions, performance issues, and security vulnerabilities [2, 3, 4]. Indeed, recent studies have shown that about 80% of enterprise security incidents originate from misconfigurations [5], and up to 65% of cloud security breaches are directly attributable to configuration mistakes [6].

A key factor contributing to the prevalence of misconfigurations is the widespread adoption of DevOps methodologies, especially Continuous Integration and Continuous Deployment (CI/CD) [7, 1]. Although these practices facilitate rapid software updates and deployments, they simultaneously increase the complexity and frequency of configuration changes [8, 9, 10]. Consequently, poorly understood interdependencies and inadequately validated changes can quickly propagate across systems, triggering widespread failures or vulnerabilities [11, 12].

Traditional troubleshooting methods typically involve manual triaging by expert administrators or rely on static decision-tree-based systems [13]. Manual approaches, though flexible, are labor-intensive, prone to human error, and difficult to scale. Static decision-tree methods [14], on the other hand, lack the adaptive capabilities necessary to handle the inherent ambiguity in user-reported symptoms and frequently fail when initial inputs are incomplete or imprecise. Additionally, existing diagnostic procedures based on standardized tests often lack discriminative power, making it difficult to precisely pinpoint problematic configuration variables (CVs) [15, 16, 17].

The initial diagnosis step typically begins with a user complaint, often ambiguous or vague, requiring iterative clarification. In conventional settings, an administrator might manually elicit additional information before preparing a trouble ticket, but this approach is neither scalable nor consistently effective. Recent advancements in large language models (LLMs) [18, 19, 20, 21] provide a promising opportunity to automate and enhance this process. Unlike traditional chatbot systems that rely on extensive databases of predefined responses, LLM-driven agents possess remarkable capabilities for generalization, requiring only limited training data to adapt effectively to diverse user inputs and conversational contexts [22, 23, 24, 25].

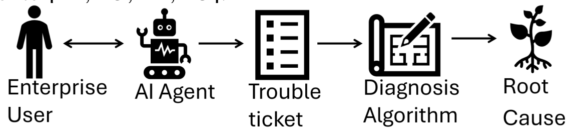


Fig. 1: Illustration of AI Agent in Diagnosis

In this paper, we present a novel self-adaptive AI agent [26, 27] that supports root-cause diagnosis of enterprise network misconfigurations [28, 29, 30]. The agent interacts with users through natural language dialogue, guided by an LLM (Mistral-7B-Instruct) [31], and uses semantic filtering to extract structured, diagnostically relevant information. This information is passed to a diagnostic backend, powered by (ConfExp), which incrementally isolates the root cause through confidence-guided test execution [32]. We describe the full workflow of our AI-agent–guided diagnostic pipeline in Section II, including symptom elicitation, CV mapping, test suggestion, ticket generation, and integration with ConfExp's iterative diagnostic engine.

Specifically, the contributions of this paper include:

- We propose an interactive AI agent that dynamically clarifies user-reported symptoms through conversational interactions, enhanced by semantic refinement techniques.
- We develop a semantic mapping approach that translates natural language symptom descriptions into structured CVs and assigns confidence scores reflecting their likelihood of being misconfigured.

Identify applicable funding agency here. If none, delete this.

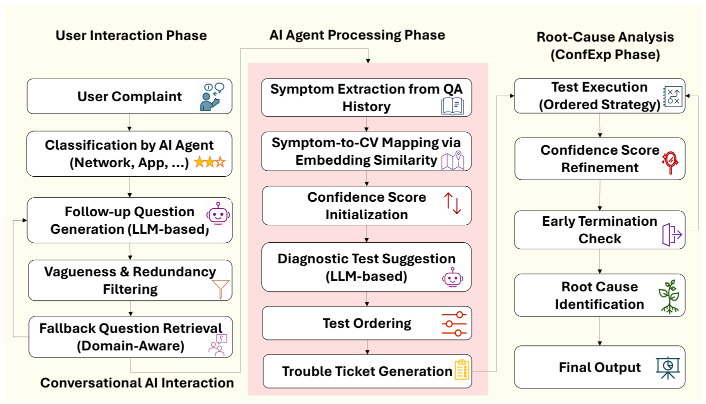

User Interaction Phase
AI Agent Processing Phase
Root-Cause Analysis (ConfExp Phase)
User Complaint
Classification by AI Agent (Network, App, ...)
Follow-up Question Generation (LLM-based)
Vagueness & Redundancy Filtering
Fallback Question Retrieval (Domain-Aware)
Conversational AI Interaction
Symptom Extraction from QA History
Symptom-to-CV Mapping via Embedding Similarity
Confidence Score Initialization
Diagnostic Test Suggestion (LLM-based)
Test Ordering
Trouble Ticket Generation
Test Execution (Ordered Strategy)
Confidence Score Refinement
Early Termination Check
Root Cause Identification
Final Output

Fig. 2: AI Agent Integrated with Misconfiguration Diagnosis.

- We propose a stepwise diagnostic method that identifies complex service and network misconfigurations using confidence-based prioritization for efficient resolution.
- We evaluate our diagnostic approach comprehensively using realistic emulations of enterprise networks, running actual enterprise-grade services, and conducting diagnostic tests commonly available to administrators, ensuring immediate practical deployability.
- We show that the AI-guided framework significantly reduces test overhead by up to 47% compared to using ConfExp alone while maintaining high accuracy (93%) and robustness even when the initial user input does not accurately reflect the underlying issue.
- We highlight two representative edge cases: (a) when the user provides the correct symptom, the AI agent's suggestions guide ConfExp to the right misconfiguration; (b) when the user provides irrelevant symptoms throughout, ConfExp still converges to the root cause using broader system-level tests and confidence refinement.

The rest of this paper is organized as follows. Section II presents an overview of our AI-driven diagnostic system and its integration with the ConfExp framework. Sections II-B through II-F describe the system components. Section III details our evaluation, and Section IV concludes.

II. System Overview

The proposed diagnostic solution integrates a conversational AI agent with an existing automated diagnostic framework, *ConfExp* [32], previously developed for systematic misconfiguration detection. An overview of ConfExp's internal mechanisms, including its confidence refinement logic and topology, is provided in Section II-A. The overarching objective is to seamlessly combine dynamic, interactive user-symptom elicitation with systematic, automated misconfiguration detection at the network and service layers. Figure 2 illustrates the overall architecture and workflow. The diagnostic pipeline consists of the following primary components:

1) Upon receiving a user's initial report, the AI agent (based on Mistral-7B-Instruct LLM) dynamically generates contextually relevant follow-up questions. An internal iterative refinement mechanism, involving semantic filtering and redundancy checks, ensures each query contributes meaningful diagnostic insight without redundancy or ambiguity. If a generated query fails to meet quality criteria, alternative fallback questions are dynamically selected to maintain dialogue efficiency.
2) The AI agent categorizes the problem into one of several domains (e.g., network issue, application server issue, or database-related problem). The agent extracts structured technical symptoms from user responses using semantically guided summarization. Ambiguous or irrelevant responses are filtered out, ensuring that only actionable, technically relevant diagnostic information is retained.
3) Extracted symptoms are semantically matched against a set of CVs. Each CV is assigned an initial confidence score indicating the likelihood of it being a source of misconfiguration. This structured output provides crucial guidance for targeted diagnostic test selection.
4) Leveraging the semantic relationships between extracted symptoms and known diagnostic commands, the AI agent generates a structured diagnostic ticket containing a prioritized list of recommended Linux diagnostic tests. These recommended tests are categorized as described in Section II-E, to guide targeted execution.
5) The structured output serves as the entry point for ConfExp's diagnostic engine, which executes targeted tests and iteratively updates CV confidence scores to identify likely root causes.

A. *Overview of ConfExp*

ConfExp [32] is a confidence-driven, test-based fault diagnosis framework developed to identify and localize misconfigurations in distributed enterprise environments. It works with realistic Enterprise network topologies that include routers, DNS servers, application and database servers, and multiple subnets. Each subnet is assumed to be overseen by a Primary Local Management Console (PLMC), with redundancy provided by a Secondary Local Management Console (SLMC). A centralized Global Management Console (GMC) coordinates fault detection and diagnosis across subnets (Figure 3).

Diagnosis in ConfExp is initiated by user complaints or observed anomalies and proceeds across two interrelated layers: service-level diagnosis, which targets application-specific issues, and network-level diagnosis, which addresses infrastructure-level faults such as routing or DNS errors. Each potentially faulty CV is assigned a confidence score in the range $[-1, 1]$, where lower scores indicate higher suspicion. At initialization, ConfExp identifies affected services and compiles a corresponding set of relevant CVs and diagnostic tests. These include widely available, non-intrusive commands such as `ping`, `traceroute`, `host`, and `netstat`. Each test is annotated with relevance scores to specific fault classes using a domain-informed Test Relevance Matrix.

Diagnosis proceeds iteratively. In each iteration, the system dynamically selects the most informative test using a strategy that considers: (i) the test's relevance to observed symptoms,

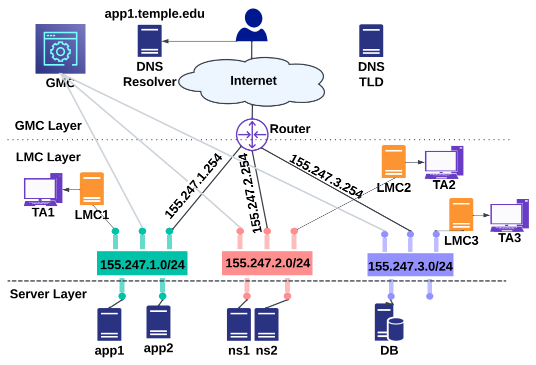


Fig. 3: Network topology used in ConfExp [33].

(ii) the number of CVs it can influence, and (iii) the current confidence distribution across CVs. Tests are prioritized to maximize diagnostic yield. After the test execution, confidence scores for associated CVs are updated. Test outcomes are used to update CV confidence levels according to the refinement rule defined in Section II-F. This includes weighted updates based on test relevance and the number of CVs associated with each test. This process continues until either the score of some CV drops below a predefined threshold (e.g., -0.6), suggesting it is a likely root cause, or no remaining tests offer meaningful diagnostic value. If all tests are exhausted without a definitive diagnosis, the CV with the lowest negative confidence is flagged for follow-up.

The PLMC continuously monitors local services and reports anomalies to the GMC. To ensure fault tolerance, it periodically synchronizes with the SLMC. If the PLMC becomes unresponsive, the SLMC autonomously assumes responsibility for diagnosis and monitoring, informing the GMC of its role change and continuing operation until the PLMC is restored. The GMC collects global state data by polling PLMC and SLMC nodes, aggregates failure reports, and uses cross-reference data to distinguish between localized faults and broader systemic misconfigurations. The global visibility provided by GMC is essential for correlating inconsistently manifested failures across subnets.[1]

Service-level diagnosis targets faults such as database connection issues or web application outages. In such cases, ConfExp focuses on CVs like port bindings, service availability, or interface mismatches. For network-level faults (e.g., DNS failures, gateway misconfigurations, or routing loops) it prioritizes diagnostic tools like `host`, `dig`, `ping`, and `traceroute`, and analyzes CVs tied to DNS, firewall, and routing configurations. A key distinction is that service tests are often executed locally by the PLMC or SLMC whereas network-level tests may span subnets and require cross-site coordination by the GMC. For example, if a service is available in one region but not in another, the GMC uses such discrepancies to infer a likely network misconfiguration.

Test execution is not fixed in order but dynamically guided by a combination of factors: current CV confidence values, test relevance, and expected information gain. This adaptive strategy allows for rather efficient diagnosis with most misconfigurations resolved within 4-5 tests and typically within 20 seconds.

[1]Although ConfExp does not implement a backup GMC, this is essential and can be added easily.

B. Dialogue and Follow-up Generation

At the core of our AI agent lies a dialogue engine designed to interactively elicit misbehavior symptoms from users through follow-up questions. The engine is implemented using the popular LLM Mistral-7B-Instruct, optimized for instruction-tuned generation. Its primary function is to produce one informative follow-up question per turn, tailored to the conversation's context and aligned with the evolving diagnostic hypothesis.

To structure the input for question generation, the agent constructs a prompt by concatenating four components: the internal instruction template I (e.g., "Ask one follow-up troubleshooting question"), the current issue classification C (such as network, application, or database), the last k conversational turns H (typically $k = 5$) to preserve context, and the user's most recent message U. Formally, we denote this as: Prompt $= \texttt{concat}(I, C, H, U)$. This structure ensures that the transformer model receives a contextually rich, task-specific input, enabling it to generate coherent and diagnostically relevant follow-up questions. The question generation process is autoregressive. At each timestep t, the model predicts the next token y_t conditioned on the preceding context and tokens, $P(y_t \mid y_{1:t-1}, X, I) = \text{softmax}(W_o \cdot \text{Transformer}(X, y_{1:t-1}))$, where X represents the embedded input prompt, and W_o is the output projection matrix. The underlying transformer architecture incorporates multi-head self-attention, feedforward layers, and rotary positional embeddings (RoPE) to preserve word order. To balance response diversity and relevance, nucleus sampling with top-$p = 0.9$ and temperature $T = 0.7$ is applied during generation. Each output is constrained to a maximum of 60 tokens, with early stopping triggered by punctuation or special tokens.

Given the variability of LLM outputs, not every generated question is diagnostically useful. To address this, the engine implements an iterative refinement process in which the model generates a series of candidate questions—each independently assessed through a multi-stage quality filter—until a diagnostically relevant and non-redundant question is accepted or a generation limit is reached. This filtering begins with vagueness detection, using both regex pattern matching and semantic similarity against a set of vague reference questions (e.g., "What else can you tell me?"). Embeddings are computed using a separate Sentence-BERT model (`all-MiniLM-L6-v2`) [34, 35], rather than Mistral, as Sentence-BERT is optimized for semantic similarity tasks and produces fixed-length embeddings suitable for cosine-based ranking. In contrast, Mistral is a decoder-only generative model, making it less suitable for efficient vector comparison across multiple candidates.

Next, lexical redundancy is evaluated using Jaccard similarity between the candidate Q_{new} and each previously asked question Q_{prev} [36]: $J(Q_{\text{new}}, Q_{\text{prev}}) = \frac{|Q_{\text{new}} \cap Q_{\text{prev}}|}{|Q_{\text{new}} \cup Q_{\text{prev}}|}$. A candidate is rejected if $J > 0.7$, indicating excessive word overlap. Structural similarity is then computed using normalized Levenshtein distance [37]: $L(Q_{\text{new}}, Q_{\text{prev}}) = \frac{D(Q_{\text{new}}, Q_{\text{prev}})}{\max(|Q_{\text{new}}|, |Q_{\text{prev}}|)}$, where $D(\cdot)$ denotes the character-level edit distance. Candidates with $L < 0.2$ are deemed structurally repetitive. Lastly, semantic redundancy is assessed using cosine similarity between Sentence-BERT embeddings [38, 34]: $S_{\text{cosine}}(Q_{\text{new}}, Q_{\text{prev}}) = \frac{Q_{\text{new}} \cdot Q_{\text{prev}}}{\|Q_{\text{new}}\| \cdot \|Q_{\text{prev}}\|}$. Any question with $S_{\text{cosine}} > 0.8$ is considered semantically overlapping and is excluded. All thresholds were selected through empirical tuning on a validation dataset to balance redundancy filtering with question diversity. Filtering occurs immediately after each generation attempt and before adding the accepted question to the dialogue history.

If no candidate passes filters after attempts, the system activates a fallback mechanism. This module selects a relevant question from a curated, domain-specific pool based on the current issue category. Each fallback candidate is also validated against user responses to avoid contradictions. For example, if a user has already indicated that other devices are unaffected, the agent will not repeat questions about multi-device impact. In our experiments across 75 fault scenarios, fallback questions were triggered in approximately 8.4% of follow-up turns (i.e., 32 out of 375 total generated questions), typically when the LLM produced vague or redundant outputs. These fallbacks preserved the dialogue flow and ensured no user turn was left unanswered.

The dialogue engine includes a dynamic scope control mechanism that determines whether to continue probing a focused diagnostic hypothesis or broaden the investigation to other potential causes. This behavior is governed by a contextual similarity metric between the newly generated question Q_{new} and the prior dialogue history H, formalized as:

$$d_{\text{expand}} = \begin{cases} \text{Expand scope}, & \text{if } S_c(Q_{\text{new}}, H) > \theta_d \\ \text{Continue focus}, & \text{otherwise} \end{cases}$$

Here, S_c denotes the cosine similarity between the new question and recent questions, computed using sentence embeddings. A high similarity score suggests that the new question repeats or stays too close to existing content, signaling diminishing returns. In such cases, the system opts to *expand scope* to cover alternative hypotheses. Conversely, when similarity is low (i.e., the new question probes a distinct aspect) the engine interprets it as diagnostic diversification and continues focusing on the same general line of inquiry. For example, if earlier questions involved DNS connectivity (e.g., “Can you access other websites?”), and the model proposes “Did you try checking with 8.8.8.8 directly?”, the high similarity prompts the system to shift topics. On the other hand, if the new question shifts toward firewall rules or local configuration (e.g., “Is your firewall blocking outbound traffic?”), the low similarity allows continued refinement within the expanded hypothesis space. Questioning continues until one of the following conditions is met: five high-quality questions have been accepted; a predefined confidence threshold is reached for one or more CVs; or the user explicitly ends the session. Once the dialogue concludes, all gathered information is passed to downstream components for symptom extraction, CV relevance scoring, and trouble ticket generation.

C. Symptom Extraction

Once the dialogue engine concludes the questioning phase, the system proceeds to extract a technically meaningful symptom set from the collected conversation. The goal of this stage is to isolate and summarize user-provided information that is directly relevant to fault localization while filtering out vague, redundant, or conversationally irrelevant content.

The extraction process operates over the structured dialogue history, which consists of alternating pairs of AI questions and corresponding user responses. For each AI–user pair, the user response is analyzed to determine whether it provides any concrete diagnostic signal. Only responses that convey technically actionable content – such as references to connectivity failures, error messages, service availability, latency, authentication issues, or system behavior anomalies – are retained. To identify symptom candidates, the AI agent uses an instruction-tuned model conditioned on the full AI–user dialogue history presented in a structured question–and–answer format. The instructions are designed to extract only concrete technical indicators while explicitly avoiding inferred or unverifiable content not mentioned by the user. Each candidate symptom is then subjected to a semantic deduplication step using the cosine similarity method described in Section II-B. Sentence embeddings are computed for all symptom entries, and pairs with high semantic overlap (similarity > 0.9) are considered redundant, only one is retained. This threshold is intentionally conservative to ensure that only near-verbatim or semantically identical descriptions (e.g., “the website doesn’t load” vs. “unable to open the webpage”) are merged.

While this may seem contradictory to the earlier scope-expansion logic, where high similarity between follow-up questions triggers broader inquiry, it serves a complementary purpose. In the dialogue phase, high similarity indicates saturation and prompts topic diversification. In contrast, during symptom extraction, the high similarity between user responses signals redundancy and justifies consolidation. This separation ensures the dialogue remains exploratory while the resulting symptom set is concise and technically focused.

To further improve precision, we apply an optional semantic filtering pass that removes low-information patterns, such as vague phrases like “something is wrong“ or “it’s not working“, unless they are accompanied by specific contextual details (e.g., “it’s not working when I connect to the VPN”). While such entries may eventually be excluded during trouble ticket generation, removing them earlier helps streamline the symptom set, reduce noise in the CV relevance scoring phase, and avoid unnecessary dilution of symptom embeddings. This preemptive filtering improves both the interpretability and semantic sharpness of the extracted symptom vector, thereby enhancing downstream confidence estimation and test prioritization.The final output of this stage is a ranked and deduplicated list of technically specific user-observed symptoms. This

Algorithm 1: Dialogue-Driven Symptom Collection

```
Input: Initial user input $u_0$
Output: Dialogue history $H$, symptom set $S$
1 Classify issue type: $c \leftarrow f_{\text{classify}}(u_0)$;
2 Initialize: $H \leftarrow [u_0]$, $q \leftarrow 0$;
3 while $q < q_{\max}$ do
4     $Q_q \leftarrow f_{\text{ask}}(c, H)$;
5     if valid($Q_q$) then
6         Append $Q_q$, user reply $A_q$ to $H$;  $q \leftarrow q + 1$;
7 $S \leftarrow f_{\text{symptoms}}(H)$;
8 return $H, S$
```

list serves as the input for the CV mapping module and guides the system's next steps in estimating the relevance and confidence of potential misconfigurations. Because this symptom set reflects both user-specific context and dynamic dialogue evolution, it allows the system to perform personalized and targeted diagnostics beyond what rule-based systems could achieve with static logs or structured templates. The user-interactive elicitation pipeline is summarized in Algorithm 1.

To better understand system behavior, we consider two edge cases: (a) when the user provides the correct symptom, the system adjusts CV confidence accordingly and converges efficiently; and (b) when the user provides only irrelevant symptoms, the system relies more heavily on system-level tests. In both cases, the confidence refinement process still leads to the correct diagnosis, albeit with more steps in case (b). These scenarios are explored further in Section III-B.

D. Mapping to Configuration Variables (CVs)

After extracting structured, semantically filtered technical symptoms, the system proceeds to estimate which CVs are most likely implicated in the reported issue. This estimation step transforms unstructured symptom narratives into a structured belief distribution over CVs, which serves as input to the test prioritization and root-cause isolation phases.

Let $S = \{s_1, s_2, \ldots, s_n\}$ denote the set of user-reported symptoms, and let $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ represent the full set of known CVs relevant to the enterprise diagnostic domain. Each CV corresponds to a network- or service-level configuration entity, such as interface IP, DNS entry, port number, default gateway, or application-specific parameter. The goal is to infer an initial confidence score for each $c_j \in \mathcal{C}$ based on its semantic alignment with the aggregated user symptom set. To map the extracted symptoms to relevant CVs, we use the cosine similarity metric as described in Section II-B. Both the aggregated symptom description and each candidate CV are embedded into a shared semantic space using a sentence embedding model (Sentence-BERT).

Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of extracted symptoms, and $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ the full CV set. We compute a combined symptom embedding vector v_S and compare it with each CV embedding v_j. The cosine similarity score σ_j quantifies semantic alignment between S and each c_j. We then assign each CV an initial confidence value:

$$\gamma_j = \begin{cases} -\alpha \cdot \sigma_j, & \text{if } \sigma_j > \theta \\ 0, & \text{otherwise} \end{cases}$$

Algorithm 2: CV Confidence Scoring

```
Input: Symptom set $S$, CV set $\mathcal{C}$
Output: Initial confidence scores $\gamma$
1 $v_S \leftarrow \text{embed}(s_1 \cup s_2 \cup \cdots \cup s_n)$;
2 Let $\mathcal{R} \subseteq \mathcal{C}$;
3 foreach $c_j \in \mathcal{R}$ do
4     $v_j \leftarrow \text{embed}(c_j)$;   $\sigma_j \leftarrow \frac{v_S \cdot v_j}{\|v_S\| \cdot \|v_j\|}$;
5     if $(\sigma_j > \theta)$ then $\gamma(c_j) \leftarrow -\alpha \cdot \sigma_j$; else $\gamma(c_j) \leftarrow 0$;
6 return $\gamma$
```

where $\theta \in (0, 1)$ is a similarity threshold and α is a tunable penalty coefficient. CVs with higher semantic similarity to the symptoms receive more negative initial scores, indicating a stronger suspicion of misconfiguration.

E. Diagnostic Test Suggestion and Categorization

The result of this process is a structured diagnostic ticket: a machine-readable object containing the issue type, extracted symptoms, initial CV confidence scores, and a categorized, prioritized list of diagnostic commands. This ticket serves as the standardized hand-off point between the AI agent and the ConfExp engine. Test generation is performed by an instruction-tuned model conditioned on a structured representation of the extracted symptoms. The model is constrained to emit only Linux-compatible, read-only commands (e.g., `ping`, `traceroute`, `curl`) that do not alter the system state. While initial outputs may include justifications, post-processing retains only the executable commands.

Each suggested command and the aggregated symptom set are embedded into a shared semantic space using a Sentence-BERT encoder. Cosine similarity between the symptom embedding and each command embedding is used to rank candidates by contextual relevance. Let $T = \{t_1, t_2, \ldots, t_k\}$ be the generated commands and $\vec{v}_S$ the embedded symptom vector. The top n commands selected for execution are $T' = \text{Top}_n \left(\left\{ \left(t_i, \cos(\vec{v}_S, \vec{t_i}) \right) \mid t_i \in T \right\} \right)$, where $n = 5$ by default and $\vec{t_i} = \text{embed}(t_i)$. Commands are then classified into three categories, AI-only, intersection, and system-only, by comparing them against the ConfExp test suite. Each AI-generated command is normalized to its base executable (e.g., ping, ifconfig) and matched against ConfExp's predefined system diagnostics. This process determines overlap: AI-only commands appear exclusively in the AI output, system-only tests are native to ConfExp but not suggested by the AI, and intersection tests are shared between both. This matching facilitates later prioritization during execution, without implying that system tests are derived from the AI agent. Let $\mathcal{A}$ and $\mathcal{S}$ be the AI and system test sets. Then, AI-only: $\mathcal{A} \backslash \mathcal{S}$, Intersection: $\mathcal{A} \cap \mathcal{S}$, and System-only: $\mathcal{S} \setminus \mathcal{A}$. Algorithm 3 summarizes test selection and confidence refinement.

F. ConfExp Integration

The AI-generated ticket is passed to ConfExp's execution engine, which performs a confidence-guided diagnostic sequence using the mechanisms outlined in Section II-A. This section describes how ConfExp prioritizes and executes tests

Algorithm 3: Test Suggestion and Confidence Update

```
Input: Symptom set $S$, initial scores $\gamma$
Output: Final ticket $\tau$, updated $\gamma$
1 Generate test set: $T \leftarrow f_{\text{tests}}(S)$;
2 Select top-$n$: $T' \leftarrow \text{Top}_n(T)$;
3 Let $\mathcal{A} \leftarrow \text{base}(T')$, $\mathcal{S} \leftarrow \text{SystemTests}$;
4 Categorize:
    AI-only $= \mathcal{A} \setminus \mathcal{S}$,   Intersection $= \mathcal{A} \cap \mathcal{S}$,   System-only $= \mathcal{S} \setminus \mathcal{A}$
    $\tau \leftarrow (c, S, T', \gamma, \text{categories})$;
5 foreach $t_i \in T'$ do
6     Run $t_i$, obtain result $r_i$;
7     foreach $c_j \in CVs(t_i)$ do
8         $\gamma(c_j) \leftarrow \gamma(c_j) + \delta(r_i, c_j)$;
9 return $\tau, \gamma$
```

Algorithm 4: Integration of AI Agent with ConfExp

```
Input: CV set $\mathcal{C}$, test sets $T_{\text{AI}}, T_{\text{Int}}, T_{\text{Sys}}$, initial scores $\gamma$, threshold $\tau$
Output: Final CV scores $\gamma$, suspected misconfigurations
1 foreach $t_i \in T_{AI} \cup T_{Int}$ do
2     Run $t_i$, obtain result $r_i \in \{\text{pass}, \text{fail}\}$;
3     Let $\text{CVs}_i \leftarrow \text{CVs}(t_i)$, $w_i \leftarrow \text{weight}(t_i)$;
4     foreach $c_j \in CVs_i$ do
5         if $(r_i = \text{pass})$ then $\gamma(c_j) \leftarrow \gamma(c_j) + w_i/|\text{CVs}_i|$;
6         else $\gamma(c_j) \leftarrow \gamma(c_j) - w_i/|\text{CVs}_i|$;
7 if $\exists! \, c_j$ such that $\gamma_j \leq \tau$ and $\forall c_k \neq c_j, \gamma_k \geq 0$ then
8     return $\gamma$, flag $c_j$ as root cause;
9 if $|\{t_i \in T_{executed} \mid r_i = pass\}| \geq \frac{|T_{executed}|}{2}$ then
10    return $\gamma$, halt: system likely healthy or false alarm;
11 foreach $t_i \in T_{Sys}$ do
12    Run $t_i$, obtain result $r_i$;
13    Update $\gamma$ using same rule as above;
14 return Final CV scores $\gamma$, all $c_j$ with $\gamma_j \leq \tau$
```

based on the ticket and how it determines when to terminate a diagnosis. Each diagnostic test t_i is associated with a subset of CVs, denoted $\text{CVs}(t_i) \subseteq \mathcal{C}$, and a diagnostic weight w_i. When a test is executed, the confidence score γ_j of each associated CV c_j is updated based on the outcome. A passing test increases the confidence score (indicating the CV is less likely to be faulty), while a failing test decreases it. This update rule, detailed in Section II-A, distributes the diagnostic signal across all CVs implicated by the test and drives the system toward convergence through accumulating evidence.

Test execution follows some order among these three types of tests: (1) AI-only tests, (2) intersection tests, and (3) system-only tests. We will evaluate the effectiveness of each of the six different orders.After executing the AI-only and intersection test phases, ConfExp checks for early termination based on two criteria. First, the system halts if there exists exactly one CV c_j such that $\gamma_j \leq \tau$ and all other CVs $c_k \in \mathcal{C} \setminus \{c_j\}$ satisfy $\gamma_k \geq 0$, where τ is a threshold confidence value (typically -0.6). This indicates that a single CV is strongly implicated, while the rest are cleared of suspicion. The value of τ has been empirically tuned to balance fault isolation accuracy and false positive suppression, as previously described in [32].

Second, ConfExp applies an early exit condition if at least half of the executed diagnostic tests have passed. Formally, if $|\{t_i \in T_{\text{executed}} \mid r_i = \text{pass}\}| \geq \frac{|T_{\text{executed}}|}{2}$, the system concludes that the environment is likely healthy and halts further diagnosis. This behavior is particularly useful for handling spurious, transient, or nonexistent faults; for example, when the user complaint does not correspond to an actual misconfiguration. In such cases, if no CV accumulates a sufficiently negative confidence score and the majority of test outcomes are non-indicative of faults, the system conservatively terminates and flags the complaint as unverifiable. This safeguard ensures diagnostic efficiency even when the input is vague, misleading, or fabricated.[2]

If neither early termination condition is satisfied, i.e., multiple CVs remain moderately suspicious or confidence values are inconclusive, the system continues with system-only tests. As outlined in Section II-A, these tests are not executed in a fixed order; instead, ConfExp dynamically selects the next test based on its diagnostic weight, association with currently suspect CVs, and its potential to shift confidence scores meaningfully. This adaptive strategy enables ConfExp to focus on tests with the greatest likelihood of resolving ambiguity and driving convergence.

Diagnosis concludes once either a CV's confidence falls below the misconfiguration threshold τ, or no remaining tests are expected to significantly influence the confidence distribution. The final diagnostic output includes updated confidence scores for all CVs, the full sequence of executed tests and their pass/fail results, and a ranked list of suspected misconfigurations to guide remediation efforts. This process is formally summarized in Algorithm 4, which reflects the integration of AI-guided and system-based testing, early exit logic, and iterative confidence refinement.

G. Related Work

Fault detection and localization have been extensively studied for both hardware and software systems. Traditional network monitoring tools like SNMP [39], NETCONF [40], and Nagios [41] primarily focus on centralized health monitoring and alerting based on device-level events. These tools, however, do not directly address detailed diagnostics at the CV level or systematically evaluate dependencies and configurations across services, which are crucial in identifying misconfigurations.

Conversational AI systems have been explored to streamline initial troubleshooting by interactively clarifying ambiguous user complaints [26, 27, 22]. These systems leverage LLMs capable of interpreting and structuring ambiguous inputs [23, 24, 21]. A typical conversational AI pipeline includes natural language understanding (NLU) to extract structured data from user inputs, dialogue state tracking (DST) to maintain context, policy learning modules (PLU) for determining subsequent actions, and natural language generation (NLG) for responses [42, 23]. Some approaches utilize self-dialogue techniques to refine extracted information [43], while retrieval-augmented generation (RAG) has been suggested to improve specialized Q&A capabilities [22]. Intent-Based

[2]An enterprise will invariably authenticate and log the reporting user; therefore, diagnosis as a DoS attack mechanism is highly unlikely.

Networking (IBN) allows expressing network intentions in natural language, significantly benefiting from advancements in LLMs [44, 45, 46]. Although somewhat similar, our approach focuses explicitly on interactive symptom elicitation combined with structured, confidence-driven diagnostic execution.

Several recent works propose automated diagnosis techniques using Bayesian inference [47, 48], entropy approximation [49], and adaptive probing [48, 50, 51]. However, these methods often rely on abstract models or predefined dependency relationships rather than dynamic, conversationally derived information. In contrast, our approach integrates interactive conversational AI with systematic, confidence-driven diagnostic tests to directly handle real-world misconfiguration diagnosis at the CV level, which, to the best of our knowledge, has not been fully explored in prior works.

III. Evaluation

We evaluate the system's ability to identify misconfigured CVs, focusing on AI-initialized confidence, test ordering, and execution strategy, with a case study illustrating confidence updates and early termination.

A. Experimental Setup

We evaluate our integrated diagnostic framework using a combination of emulated enterprise environments and LLM inference deployed on a high-performance infrastructure. To simulate realistic IT environments, we use the SEED emulator [52], which provisions DNS, web servers, databases, and application services inside Docker-based containers connected through configurable subnets and routing domains. Hosts are configured with realistic system files, service dependencies, and monitoring agents. Misconfigurations are introduced by altering network settings (e.g., subnet masks, gateways), DNS records, firewall rules, port bindings, or authentication policies. Note that while the misconfigurations are necessarily deliberate for evaluation purposes, none of the monitoring and diagnosis procedures have any knowledge of or dependence on the injected fault. However, the user-reported problems must refer to an abstractly described problem resulting from the misconfiguration.To reflect real-world usage, each misconfiguration is paired with a natural language complaint written to mimic plausible end-user input. These complaints are derived by observing the operational impact of the misconfiguration (e.g., "I can't reach any websites," or "The app fails to connect to the database") and avoid revealing internal fault details. The AI agent processes each complaint through interactive questioning, extracts symptoms, and generates test recommendations that feed into the ConfExp engine.

LLM-based components, including Mistral-7B-Instruct for question generation and Sentence-BERT for semantic similarity, are deployed on the ACES high-performance cluster at Texas A&M University. We use Intel Data Center GPU Max 1100 (PVC) nodes with SLURM-based container orchestration, following ACES best practices [53]. Each diagnostic session runs in isolation, and all test execution occurs within the emulated environment. We evaluate the system across 75 curated fault scenarios spanning diverse categories of misconfigurations, including 25 network-layer issues (such as misconfigured IP addresses, broken gateways, and DNS failures), 25 application/server misconfigurations (such as incorrect URLs, unreachable ports, and crashed services), 15 database-related faults (including blocked ports, authentication failures, and unreachable database endpoints), and 10 multi-layer composite faults that span both infrastructure-level (e.g., IP, DNS, routing, or firewall rules) and application-layer (e.g., server responsiveness, SSL, or port bindings) components. A breakdown of fault types is provided in Table I.

TABLE I: Distribution of Fault Scenarios by Category.

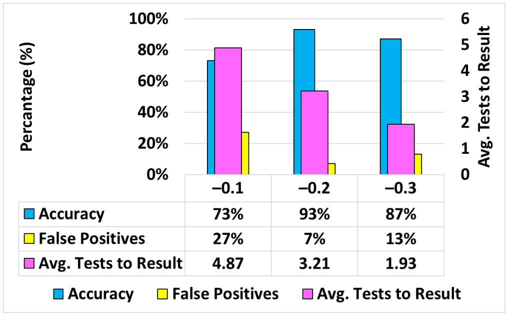
Fault Type	**Number of Scenarios**
Network-layer misconfigurations	25
Application/server misconfigurations	25
Database connectivity/configuration errors	15
Multi-layer composite faults	10
Total	**75**

For each scenario, the system turns complaints into diagnostic tickets, executes tests via ConfExp, and tracks CV confidence, coverage, and convergence. Results are in Section III-B.

B. Experimental Results

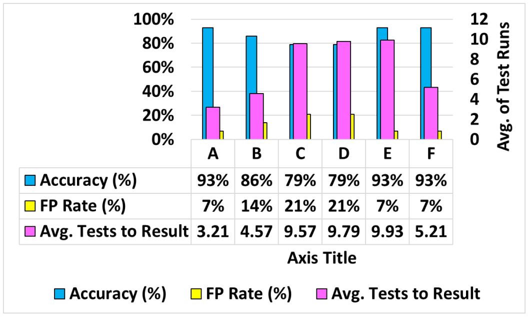
We evaluated the diagnostic performance of our system across all emulated enterprise fault scenarios, each involving natural language user complaints that were converted into structured trouble tickets, test recommendations, and automated diagnostic execution via ConfExp. Our analysis focuses on how the AI-assigned initial confidence levels (γ_j) and the sequence in which tests are executed affect the system's ability to efficiently and accurately localize misconfigurations.

To initialize CV confidence values, the system uses semantic similarity scores between extracted user symptoms and candidate CVs, scaled by a tunable factor α. Specifically, for each relevant CV c_j, the initial confidence is computed as $\gamma_j = -\alpha \cdot \sigma_j$, where σ_j is the cosine similarity score. We tested three values of $\alpha \in \{0.1, 0.2, 0.3\}$ to study how strongly initial suspicion influences diagnostic behavior. Each configuration was evaluated using three key metrics: (1) accuracy: the percentage of scenarios where the correct CV was ultimately identified (including ties); (2) false positives: the percentage of cases where an incorrect CV crossed the threshold before the correct one; and (3) average number of tests to result. Although we do not observe conventional false negatives, i.e., cases where the correct misconfiguration is missed entirely, even in scenarios where the user's initial symptom description does not align with the actual fault, the system still successfully identifies the true faulty CV. However, doing so requires more diagnostic tests and longer execution times, as the initial confidence estimates are skewed toward irrelevant variables. In these cases, the system recovers by leveraging its confidence refinement loop and broader system-defined tests, which eventually suppress incorrect hypotheses and converge on the true root cause. Thus, while accuracy remains high, test efficiency degrades under incorrect user input, as detailed in Section III-B and illustrated in Figure 7.



	−0.1	−0.2	−0.3
Accuracy	73%	93%	87%
False Positives	27%	7%	13%
Avg. Tests to Result	4.87	3.21	1.93

Fig. 4: Impact of coefficient α on CV diagnosis performance.



	A	B	C	D	E	F
Accuracy (%)	93%	86%	79%	79%	93%	93%
FP Rate (%)	7%	14%	21%	21%	7%	7%
Avg. Tests to Result	3.21	4.57	9.57	9.79	9.93	5.21

Fig. 5: Diagnostic performance across six test strategies.

TABLE II: Description of Test Ordering Strategies

Strategy	Execution Order
A	AI Agent–Only $\rightarrow$ Intersection $\rightarrow$ System-Only
B	Intersection $\rightarrow$ AI Agent–Only $\rightarrow$ System-Only
C	System-Only $\rightarrow$ AI Agent–Only $\rightarrow$ Intersection
D	System-Only $\rightarrow$ Intersection $\rightarrow$ AI Agent–Only
E	Intersection $\rightarrow$ System-Only $\rightarrow$ AI Agent–Only
F	AI Agent–Only $\rightarrow$ System-Only $\rightarrow$ Intersection

As shown in Figure 4, a moderate value of $\alpha = 0.2$ yielded the best trade-off: 93% accuracy, only 7% false positives, and an average of 3.21 tests. A lower value ($\alpha = 0.1$) led to weaker initial suspicion, resulting in more tests (4.87 on average) and a 27% false positive rate due to slow convergence. Conversely, an aggressive setting ($\alpha = 0.3$) shortened diagnostics (1.93 tests) but increased false positives to 13%, as the system prematurely committed to incorrect CVs. We limited the range of α values to these three settings based on prior empirical tuning. Values below 0.1 delayed diagnosis too significantly to be practical, while higher magnitudes (> 0.3) consistently caused instability and misclassification during pilot runs. Thus, the selected range reflects a practical spectrum of initialization strengths that preserve test-driven refinement without biasing the system toward early misjudgment.

Beyond confidence initialization, diagnostic performance also depends on the sequencing of test execution. We evaluated six distinct test ordering strategies that vary the execution order of AI Agent–Only, Intersection, and System-Only test categories. As summarized in Table II, these strategies reflect different prioritization philosophies, ranging from symptom-driven (AI-first) to purely system-defined approaches.

Figure 5 presents the aggregate performance of each strategy across all 75 fault scenarios, measured by accuracy, false positive rate (FP), and average tests to result. Strategy A, prioritizing symptom-aligned (AI) tests, achieved the best overall performance: 92.86% accuracy, 7.14% false positives, and only 3.21 tests per scenario. In contrast, Strategies C and D, which start with System-Only tests, performed significantly worse—requiring over 9 tests on average and exhibiting the highest FP rates (21.43%). Intermediate strategies, such as B and E (which front-load consensus or mixed tests), showed moderate gains in accuracy but failed to reduce test overhead. Notably, Strategy F matched Strategy A in accuracy but required 60% more tests, demonstrating that deferred consensus testing delays early exits without improving results. These findings underscore the value of aligning initial test execution with user-expressed symptoms to enable early convergence and minimize unnecessary probes.

AI-Generated Trouble Ticket (VPN Issue)

Timestamp: 2025-04-03 21:18:45
Issue Type: Network Connectivity
Symptoms Reported:

- VPN drops frequently when connected to local Wi-Fi
- VPN is stable when using LTE
- Other network services remain unaffected
- Wi-Fi signal is weak, but connected

Initial Confidence Levels:

- `mtu`: –0.19, `interface`: –0.15, `ip`: –0.10

AI-Generated Diagnostic Tests:

- ⋆`ifconfig`, ⋆`ping -s 1400 8.8.8.8`, ⋆`mtr 8.8.8.8`, ⋆`ip route`

Test Categorization:

- **AI–Only:** ⋆`mtr 8.8.8.8`, ⋆`ping -s 1400 8.8.8.8`
- **Intersection:** ⋆`ifconfig`, ⋆`ip route`
- **System-Only:**
  - Name Resolution: ⋆`host google.com`
  - Reachability: ⋆`traceroute 8.8.8.8`, ⋆`nmap 8.8.8.8`, ⋆`ip addr`, ⋆`netstat -i`
  - Firewall and Services: ⋆`iptables -L -v`, ⋆`netstat -tulpn`, ⋆`telnet vpn.server.com 1194`

Fig. 6: Example of an AI-generated trouble ticket[1].

Case Study: To illustrate how test sequencing and confidence dynamics influence diagnostic outcomes, we present a scenario involving intermittent VPN connectivity. The user reported frequent VPN drops over Wi-Fi, while connections over LTE remained stable. Based on semantic similarity to the complaint, the AI agent initialized three CVs, `mtu`, `interface`, and `ip`, with confidence values of $\gamma_{\texttt{mtu}} = -0.19$, $\gamma_{\texttt{interface}} = -0.15$, and $\gamma_{\texttt{ip}} = -0.1$. The misconfiguration involved the `mtu` parameter, which was eventually isolated through targeted test execution. The structured diagnostic ticket generated by the AI agent, detailing initial scores, test suggestions, and categorization, is shown in Figure 6, and served as input to the ConfExp diagnostic engine. During diagnosis, confidence values were updated according to the rule in Section II-A. For instance, a test like `ping -s 1400`, which exclusively targets `mtu`, had a larger per-CV impact than a test like `ifconfig`, which affects multiple variables.

TABLE III: Strategy A

Test	Result	New CLs
ifconfig	Fail	interface = –0.35, mtu = –0.39, ip = –0.30
ping -s 1400	Fail	**mtu = –0.6**

In Strategy A, the system began with AI-suggested tests that were highly aligned with the user's symptoms (Table III). The `ifconfig` test, which failed, reduced the confidence of all three CVs, and the subsequent failure of `ping -s 1400` drove the confidence of `mtu` precisely to the threshold value of -0.6. As the other CVs remained above zero, the early termination condition was satisfied, and the diagnosis concluded after only two tests. Strategy F (Table VI, which followed the same test sequence, also reached early termination with identical efficiency.

TABLE IV: Strategy B

Test	Result	New CLs
ip route	Pass	ip = 0.0, interface = –0.05
ifconfig	Fail	interface = –0.25, mtu = –0.39, ip = –0.20
ping -s 1400	Fail	**mtu = –0.6**
mtr	Fail	ip = –0.40
netstat -i	Pass	interface = –0.05
traceroute	Fail	ip = –0.50

Strategy B (Table IV) took a slightly different path, beginning with the intersection test `ip route`, which passed and increased the confidence of `ip` and `interface`. Although this delayed the degradation of those variables, the following two tests (`ifconfig` and `ping -s 1400`) succeeded in pushing `mtu` to the threshold. Thus, this strategy also triggered early termination, albeit with an extra test step.

In contrast, Strategies C, D, and E (Table VI) began with system-defined tests unrelated to the user's symptoms. These sequences involved broader diagnostics like `host`, `nmap`, and `ip addr`, which primarily targeted `dns`, `port`, or general interface properties. As a result, confidence in `mtu` declined more gradually and never reached the threshold of -0.6, even after 12 diagnostic steps. However, since `mtu` ended with the most negative confidence among all CVs (approximately -0.5), it was still selected as the most likely root cause. This fallback mechanism enables convergence even when the termination condition is not met, but at the cost of additional test overhead and slower resolution compared to AI-prioritized strategies.

This case study demonstrates two core benefits of the proposed approach. First, semantically driven initialization allows the system to front-load suspicion onto symptom-relevant variables. Second, test ordering is crucial for diagnostic efficiency: AI-prioritized strategies apply early, high-impact probes that rapidly isolate the root cause and minimize overhead, whereas system-centric approaches dilute diagnostic power across less relevant tests and delay convergence.

To assess the robustness of our diagnostic system under imperfect conditions, we simulated a scenario where the user's report described symptoms unrelated to the true underlying issue. Instead of describing intermittent VPN drops, the user

[1]Newlines compressed to save space, "*" indicates the start of a diagnostic command.

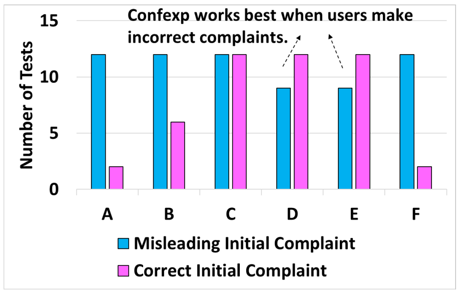


Fig. 7: Impact of Initial Complaints on Diagnostic Tests.

complaint indicated general performance issues after VPN reconnect (e.g., "VPN slow after reconnect, website slow"). Consequently, the AI agent initialized incorrect CVs (`interface` = -0.15, `port` = -0.1) and suggested irrelevant tests targeting `ip`, `url`, and `port`, missing the actual root cause (`mtu`). Despite this faulty initialization, the system still correctly diagnosed the misconfigured `mtu` across all strategies, although with increased overhead. For example, under Strategy A, the AI-issued tests (`curl`, `ping`, `traceroute`) initially boosted confidence in `ip` and `url` while leaving `mtu` untouched. It wasn't until deeper into the system-defined test phase that tests like `ifconfig` and `ip addr` revealed the actual misconfiguration. All strategies eventually converged on `mtu`, but none achieved early termination because the incorrect initialization caused other CVs to accumulate positive scores, violating the termination condition.

Figures 8(b) and 7 illustrate the system's behavior in two edge cases: (a) when the user provides accurate symptoms, and (b) when the user provides misleading or unrelated symptoms. In case (a), Strategy A required only 2 tests to isolate the fault; in case (b), the same strategy needed 12 tests due to incorrect initial confidence estimates. Despite the added overhead, the system consistently converged to the correct root cause (`mtu`) in all cases. This underscores two core strengths: (1) symptom-aware initialization improves efficiency and AI-informed strategies (e.g., A, F) perform well, and (2) confidence-guided testing enables robust diagnosis even under noisy or incorrect user complaints.

To further analyze the role of semantic alignment in AI-guided initialization, we repeated the VPN diagnosis experiments under a fixed confidence setup. In this setting, each CV that matched a user-reported symptom was assigned a constant confidence value of $\gamma_j = -0.2$, without cosine similarity–based weighting. This simplification removes fine-grained distinctions between closely and loosely aligned symptoms. Figure 8(c) summarizes the diagnostic outcomes for correct initial user complaints. Most strategies remained correct but saw no benefit in test efficiency. Notably, Strategy B misidentified `ip` as the faulty variable in the correct complaint case, despite the ground truth being `mtu`. This occurred because all matched CVs were initialized equally, allowing irrelevant variables to dominate after early false test signals. Figure 8(a) further visualizes the number of tests required in each strategy. The bars reveal that removing semantic scaling does not

TABLE V: Strategies C, D, and E.

Strategy C

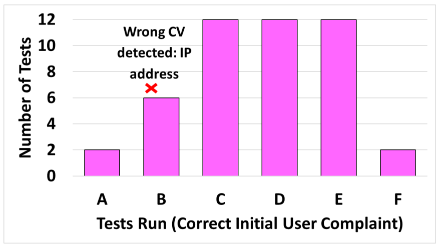
Test	Result	New CLs
host	Pass	dns = +0.2
traceroute	Fail	ip = –0.20
nmap	Pass	ip = 0.0, port = +0.2
ip addr	Pass	interface = –0.05, mtu = –0.09
iptables	Pass	—
telnet	Pass	port = +0.4
netstat -i	Pass	interface = +0.15
netstat -tulpn	Pass	port = +0.6
ifconfig	Fail	interface = –0.05, mtu = –0.29, ip = –0.20
ping -s 1400	Fail	**mtu = –0.5**
mtr	Fail	ip = –0.20
ip route	Pass	ip = –0.10, interface = +0.05

Strategy D

Test	Result	New CLs
host	Pass	dns = +0.2
traceroute	Fail	ip = –0.20
nmap	Pass	ip = 0.0, port = +0.2
ip addr	Pass	interface = –0.05, mtu = –0.09
iptables	Pass	—
telnet	Pass	port = +0.4
netstat -i	Pass	interface = +0.15
netstat -tulpn	Pass	port = +0.6
ip route	Pass	ip = +0.1, interface = +0.25
ifconfig	Fail	interface = +0.05, mtu = –0.3, ip = –0.10
ping -s 1400	Fail	**mtu = –0.5**
mtr	Fail	ip = –0.10

Strategy E

Test	Result	New CLs
ip route	Pass	ip = 0.0, interface = –0.05
host	Pass	dns = +0.2
traceroute	Fail	ip = –0.10
nmap	Pass	ip = +0.1, port = +0.2
ip addr	Pass	interface = 0.05, mtu = 0.01
iptables	Pass	—
telnet	Pass	port = +0.4
netstat -i	Pass	interface = +0.25
netstat -tulpn	Pass	port = +0.6
ifconfig	Fail	interface = +0.05, mtu = –0.2, ip = 0.0
ping -s 1400	Fail	**mtu = –0.4**
mtr	Fail	ip = –0.10



Strategy	Misleading Complaint		Correct Complaint	
	Tests	CV	Tests	CV
A	12	mtu	2	mtu
B	12	mtu	6	mtu
C	12	mtu	12	mtu
D	9	mtu	12	mtu
E	12	mtu	2	mtu
F	9	mtu	12	mtu

Strategy	Tests Fixed	Tests Scaled	CV Scaled	CV Fixed
A	12	2	mtu	mtu
B	12	6	**ip**	mtu
C	12	12	mtu	mtu
D	9	12	mtu	mtu
E	12	2	mtu	mtu
F	9	12	mtu	mtu

Fig. 8: (a) Diagnosis Performance Without Semantic Similarity (Left). (b) Effect of Misleading vs. Correct User Complaints (Middle). (c) Impact of Cosine-Scaled vs. Fixed Initial Confidence Values (Right).

TABLE VI: Strategy F

Test	Result	New CLs
ifconfig	Fail	interface = –0.35, mtu = –0.39, ip = –0.30
ping -s 1400	Fail	**mtu = –0.6**

always increase the test count but can compromise diagnostic precision, especially when initial test results create misleading trends. Strategy A and F remained efficient and accurate, while B was particularly vulnerable to misclassification.

In addition to automated comparisons, we evaluated our system against manual diagnosis by human experts, since prior abstract fault models do not directly handle CVs or realistic tools like `traceroute` and `ping`. Five experienced administrators were given the same initial issues reported to ConfExp+AI Agent without access to follow-up AI interactions. We simulated their diagnostic flowcharts to count required tests. Table VII summarizes these results across five realistic misconfiguration scenarios. ConfExp+AI required significantly fewer tests (18 total) compared to standalone ConfExp (30) and human experts (average 36.4). Particularly, in Scenario 3 (firewall misconfiguration), ConfExp+AI identified the issue with only 3 tests, versus up to 15 tests by human experts. We assessed efficiency using two metrics: *Necessity* (minimum required tests versus those used) and *Coverage* (correct identification regardless of test count). ConfExp+AI achieved 100% coverage across all scenarios, consistently outperforming manual diagnosis in both efficiency and accuracy.

TABLE VII: Diagnostic Test Counts and Metrics by Scenario.

Approach / Metric	Scen. 1	Scen. 2	Scen. 3	Scen. 4	Scen. 5	Total
Expert 1	2	9	15	8	8	42
Expert 2	2	9	10	8	5	34
Expert 3	2	11	9	6	8	36
Expert 4	2	9	8	12	4	35
Expert 5	2	9	8	12	4	35
ConfExp	2	8	4	11	5	30
ConfExp+AI	2	3	3	5	5	**18**
Necessity %	100	100	80	100	83.3	92.7
Coverage %	100	100	100	100	100	100

IV. Conclusion

We introduced an AI-driven framework that integrates large language models with a confidence-based diagnostic engine (ConfExp) to automate trouble ticket generation and misconfiguration diagnosis in enterprise environments. The system converts natural language complaints into structured symptom sets, estimates likely faulty CVs, and generates targeted test plans. Evaluation across 75 fault scenarios showed that the system achieved 93% accuracy while reducing diagnostic tests by up to 47% compared to using ConfExp alone. It remained robust even when user input was vague or misleading, and consistently outperformed manual diagnosis by experts in both efficiency and completeness. Future work includes integrating retrieval-augmented generation (RAG) to leverage past diagnostic sessions, incorporating service-dependency models for correlated faults, and supporting multi-fault diagnosis through parallel test paths.

References

[1] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[2] Tianyin Xu and Yuanyuan Zhou. Systems approaches to tackling configuration errors: A survey. *ACM Computing Surveys (CSUR)*, 47(4):70, 2015.

[3] Suranjith Ariyapperuma and Chris J Mitchell. Security vulnerabilities in dns and dnssec. In *The Second International Conference on Availability, Reliability and Security (ARES'07)*, pages 335–342. IEEE, 2007.

[4] Asadullah Shaikh, Bhavika Pardeshi, and Faraz Dalvi. Overcoming threats and vulnerabilities in dns. In *Proceedings of the 3rd International Conference on Advances in Science & Technology (ICAST)*, 2020.

[5] Gartner. Gartner Identifies the Top Cybersecurity Trends for 2023, 2023. Accessed: 2025-03-07.

[6] Palo Alto Networks. Critical Cloud Misconfigurations in AWS, 2019. Accessed: 2025-03-07.

[7] Sam Newman. *Building microservices*. " O'Reilly Media, Inc.", 2021.

[8] Misconfiguration brings down entire .se domain in sweden. URL http://www.circleid.com/posts/misconfiguration_brings_down_entire_se_domain_in_sweden, 2009. [Online; accessed 3-July-2023].

[9] Martyn Williams and Ashlee Vance. Microsoft takes blame for web site access failures. URL http://www.computerworld.com/article/2590639/networking/microsoft-takes-blame-for-web-site-access-failures.html, 2001. [Online; accessed 3-July-2023].

[10] Apple blames itunes outage on dns error. what does that mean? URL https://www.csmonitor.com/Technology/2015/0311/Apple-blames-iTunes-outage-on-DNS-error.-What-does-that-mean, 2015. [Online; accessed 3-July-2023].

[11] Salman Baset et al. Usable declarative configuration specification and validation for applications, systems, and cloud. In *ACM/IFIP/USENIX Middleware Conference*, pages 29–35, 2017.

[12] Ching-Huang et al. Lin. A study and implementation of vulnerability assessment and misconfiguration detection. In *2008 IEEE Asia-Pacific Services Computing Conference*, pages 1252–1257. IEEE, 2008.

[13] Zahedi Azam, Md Motaharul Islam, and Mohammad Nurul Huda. Comparative analysis of intrusion detection systems and machine learning-based model analysis through decision tree. *IEEE Access*, 11:80348–80391, 2023.

[14] IS Saeh and A Khairuddin. Decision tree for static security assessment classification. In *2009 International Conference on Future Computer and Communication*, pages 681–684. IEEE, 2009.

[15] Weili Wang, Lun Tang, Chenmeng Wang, and Qianbin Chen. Real-time analysis of multiple root causes for anomalies assisted by digital twin in nfv environment. *IEEE transactions on network and service management*, 19(2):905–921, 2022.

[16] Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *OSDI*, volume 10, pages 1–14, 2010.

[17] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. A general approach to network configuration analysis. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 469–483, 2015.

[18] Wanwei He, Yinpei Dai, Min Yang, Jian Sun, Fei Huang, Luo Si, and Yongbin Li. Unified dialog model pre-training for task-oriented dialog understanding and generation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 187–200, 2022.

[19] Pengyu Zhao, Zijian Jin, and Ning Cheng. An in-depth survey of large language model-based artificial intelligence agents. *arXiv preprint arXiv:2309.14365*, 2023.

[20] Niels LM van Adrichem, Norbert Blenn, Antonio Reyes Lúa, Xin Wang, Muhammad Wasif, Ficky Fatturrahman, and Fernando A Kuipers. A measurement study of dnssec misconfigurations. *Security Informatics*, 4:1–14, 2015.

[21] Bibhu Dash. Zero-trust architecture (zta): Designing an ai-powered cloud security framework for llms' black box problems. *Available at SSRN 4726625*, 2024.

[22] Simone Alghisi, Massimo Rizzoli, Gabriel Roccabruna, Seyed Mahed Mousavi, and Giuseppe Riccardi. Should we fine-tune or rag? evaluating different techniques to adapt llms for dialogue. *arXiv preprint arXiv:2406.06399*, 2024.

[23] Yujie Feng, Zexin Lu, Bo Liu, Liming Zhan, and Xiao-Ming Wu. Towards llm-driven dialogue state tracking. *arXiv preprint arXiv:2310.14970*, 2023.

[24] Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. A survey on recent advances in llm-based multi-turn dialogue systems. *arXiv preprint arXiv:2402.18013*, 2024.

[25] Nguyen Van Tu, Jae-Hyoung Yoo, and James Won-Ki Hong. Towards intent-based configuration for network function virtualization using in-context learning in large language models. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pages 1–8. IEEE, 2024.

[26] Pradnya Kulkarni, Ameya Mahabaleshwarkar, Mrunalini Kulkarni, Nachiket Sirsikar, and Kunal Gadgil. Conversational ai: An overview of methodologies, applications & future scope. In *2019 5th International conference on computing, communication, control and automation (ICCUBEA)*, pages 1–7. IEEE, 2019.

[27] Shalini Chandra, Anuragini Shirish, and Shirish C Srivastava. To be or not to be... human? theorizing the role of human-like competencies in conversational artificial intelligence agents. *Journal of Management Information*

Systems, 39(4):969–1005, 2022.

[28] Mei Zhang, ZT Li, Boutaib Dahhou, Michel Cabassud, and C Volosencu. Root cause analysis of actuator fault. In *Actuators*, page 131. IntechOpen, 2018.

[29] Gustavo V Maia, Thiago M Coutinho, Eduardo B Gonçalves, Gustavo RL Silva, Eduardo MAM Mendes, Marcelo MAM Mendes, Sandro R Caetano, Gustavo M Mitt, and Antonio P Braga. One class density estimation approach for fault detection and rootcause analysis in computer networks. *Journal of Network and Systems Management*, 30(4):69, 2022.

[30] Chia-Cheng Yen, Wenting Sun, Hakimeh Purmehdi, Won Park, Kunal Rajan Deshmukh, Nishank Thakrar, Omar Nassef, and Adam Jacobs. Graph neural network based root cause analysis using multivariate time-series kpis for wireless networks. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE, 2022.

[31] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mistral 7b. *arXiv preprint*, arXiv:2310.06825, 2023.

[32] Negar Mohammadi Koushki, Ibrahim El-Shekeil, and Krishna Kant. Confexp: Root-cause analysis of service misconfigurations in enterprise systems. *JNSM*, 33(2):1–31, 2025.

[33] Negar Mohammadi, Sanjeev Sondur, and Krishna Kant. Automated configuration for agile software environments. *Proc. of IEEE Cloud*, July 2022.

[34] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[35] Chen Yin and Zixuan Zhang. A study of sentence similarity based on the all-minilm-l6-v2 model with "same semantics, different structure" after fine tuning. In *2024 2nd International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2024)*, pages 677–684. Atlantis Press, 2024.

[36] Sujoy Bag, Sri Krishna Kumar, and Manoj Kumar Tiwari. An efficient recommendation generation using relevant jaccard similarity. *Information Sciences*, 483:53–64, 2019.

[37] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.

[38] Peipei Xia, Li Zhang, and Fanzhang Li. Learning similarity with cosine similarity ensemble. *Information sciences*, 307:39–52, 2015.

[39] Jeff Case, Mark Fedor, Martin Schoffstall, and James Davin. Simple network management protocol (SNMP). *IETF RFC*, 1157, 1990.

[40] Rob Enns. Netconf configuration protocol. *IETF RFC*, 4741, 2006.

[41] Wolfgang Barth. *Nagios: System and Network Monitoring*. No Starch Press, SF, CA, 2008.

[42] Lang Cao. Diaggpt: An llm-based chatbot with automatic topic management for task-oriented dialogue. *arXiv preprint arXiv:2308.08043*, 2023.

[43] Dennis Ulmer, Elman Mansimov, Kaixiang Lin, Justin Sun, Xibin Gao, and Yi Zhang. Bootstrapping llm-based task-oriented dialogue agents via self-talk. *arXiv preprint arXiv:2401.05033*, 2024.

[44] Engin Zeydan and Yekta Turk. Recent advances in intent-based networking: A survey. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5. IEEE, 2020.

[45] Nikos Leivadeas et al. A survey on intent-based networking. *IEEE Communications Surveys & Tutorials*, 24(2):1234–1256, 2022.

[46] Matthias Falkner and John Apostolopoulos. Intent-based networking for the enterprise: A modern network architecture. *Communications of the ACM*, 65(11):70–78, 2022.

[47] Irina Rish, Mark Brodie, Natalia Odintsova, Sheng Ma, and Genady Grabarnik. Real-time problem determination in distributed systems using active probing. In *IEEE/IFIP NOMS*, volume 1, pages 133–146, 2004.

[48] Maitreya Natu and Adarshpal S Sethi. Probe station placement for fault diagnosis. In *IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*, pages 113–117. IEEE, 2007.

[49] Alice X Zheng and Irina Rish. Efficient test selection in active diagnosis via entropy approximation. *arXiv preprint arXiv:1207.1418*, 2012.

[50] Deepak Jeswani et al. Adaptive monitoring: application of probing to adapt passive monitoring. *JNSM*, 23:950–977, 2015.

[51] Lu Lu et al. A new fault detection method for computer networks. *Reliability Engineering & System Safety*, 114:45–51, 2013.

[52] Wenliang Du and Honghao Zeng. The seed internet emulator and its applications in cybersecurity education, 2022.

[53] Zhan He and William Brashear. AI/ML Workflows on ACES Accelerators. https://hprc.tamu.edu/files/aces24/AI_ML_on_ACES_ACES_Workshop_2024.pdf, July 2024. Presented at the ACES Workshop 2024, Providence, RI.