JFTA-Bench: Evaluate LLM's Ability of Tracking and Analyzing Malfunctions Using Fault Trees

Anonymous ACL submission

Abstract

In the maintenance of complex systems, fault trees are used to locate problems and provide targeted solutions. To enable fault trees stored as images to be directly processed by large language models, which can assist in tracking and analyzing malfunctions, we propose a novel textual representation of fault trees. Building on it, we construct a benchmark for multi-turn dialogue systems that emphasizes robust interaction in complex environments, evaluating a model's ability to assist in malfunction localization, which contains 3130 entries and 40.75 turns per entry on average. We train an end-to-end model to generate vague information to reflect user behavior and introduce long-range rollback and recovery procedures to simulate user error scenarios, enabling assessment of a model's integrated capabilities in task tracking and error recovery, and Gemini 2.5 pro archives the best performance.

1 Introduction

Fault Tree Analysis (FTA) is a top-down, deductive failure analysis methodology that has been widely adopted in the maintenance of complex systems for fault localization and decision support(Lee et al., 2009) . Starting from a predefined system-level top event, FTA models the causal relationships between failures using Boolean logic, primarily through AND and OR gates, and progressively decomposes system failures into basic events such as component-level malfunctions(Ruijters and Stoelinga, 2015). Subsequent extensions, including Dynamic Fault Trees(Karanki et al., 2009), further enrich the expressive power of fault trees by introducing additional logic gates and temporal dependencies, enabling more fine-grained reasoning in realistic industrial scenarios.

In parallel, large language models (LLMs), represented by systems such as GPT-5(Singh et al., 2025), Gemini 3 pro(Gemini Team, Google, 2025) and DeepSeek-V3.2(DeepSeek-AI, 2025), have achieved rapid advances in recent years . With strong capabilities in in-context learning (Dong et al., 2024), recent research on LLM-based agents(Luo et al., 2025) further highlights their applicability in complex decision-making and interactive problem-solving tasks, suggesting promising opportunities for assisting fault tree–based diagnosis. To support both professional technicians and non-expert users in efficiently locating malfunctions in complex systems without requiring manual exploration of extensive fault trees, we leverage LLMs to assist and streamline the diagnosis workflow. However, existing fault tree standards and datasets are predominantly image-based(Vesely et al., 1981), whereas most open-source and commercial LLMs are pretrained primarily on textual corpora(Zhao et al., 2026). Although multimodal models attempt to bridge this gap by projecting visual features into textual embedding spaces via connector-based alignment(Liu et al., 2023b), such approaches inevitably introduce information loss and limit precise structural reasoning(Feng et al., 2025; Liu et al., 2023a).

To address this mismatch, we propose a structured, extensible, and model-friendly textual representation of fault trees. By organizing different logic gates through explicit structural compositions and enabling cross-branch references, our representation supports faithful textual encoding of complex fault trees beyond simple tree structures. It strikes a balance between natural language expressiveness and formal structural rigor.

Building on this representation, we construct a benchmark to evaluate the ability of LLMs to learn and reason over fault trees through in-context learning alone. We collect and process a Chinese-language dataset of 126 fault trees from diverse domains, with an average of 140 nodes per tree. By introducing cross-branch links, the resulting structures reflect realistic, non-trivial diagnostic

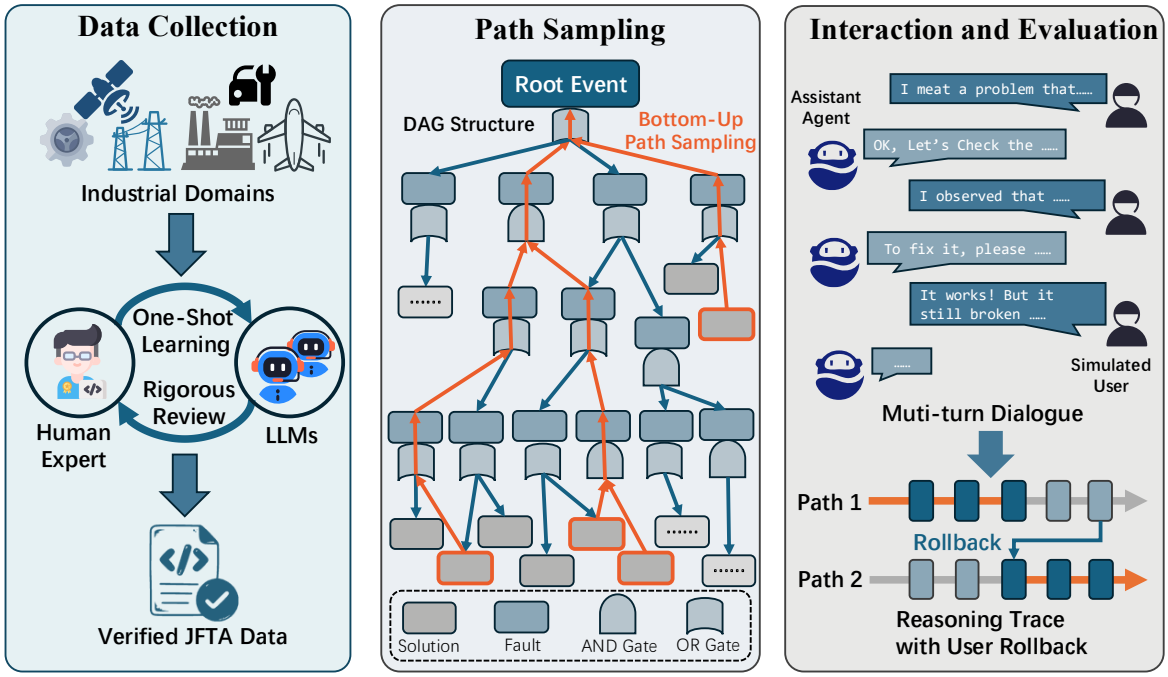

Figure 1: The left panel illustrates the Human-in-the-loop data collection process. The middle panel depicts the path sampling procedure capable of handling DAG structures. The right panel demonstrates the multi-turn evaluation framework featuring user rollback scenarios.

scenarios. From these fault trees, we sample 3,130 distinct fault path groups and organize them into multiple difficulty levels. To further approximate real-world conditions, we explicitly model user errors during interaction. We introduce long-range state rollback and recovery procedures, where users may correct or revise earlier statements in the dialogue history. This design enables systematic evaluation of a model's integrated capabilities in task tracking, consistency maintenance, and error recovery across extended multi-turn conversations.

The benchmark evaluates whether a model can correctly localize all root causes and propose appropriate solutions within a limited number of interaction turns. According to our evaluation, Gemini 2.5 pro (Comanici et al., 2025a) achieves the best overall performance, successfully passing 53.76% of the test cases by correctly identifying all malfunctions. Among open-source models, DeepSeek-V3.2 performs the strongest, reaching a success rate of 41.40%. Further analysis indicates that model failures are primarily attributable to deficiencies in planning capability

To simulate realistic and consistent user behavior in complex interactive settings, we select 100 fault trees from the dataset to train a user model that supports multi-turn dialogue evaluation. The trained user generates vague, uncertain, and occasionally biased information, better reflecting real user characteristics and potential adversarial behaviors. The remaining 26 fault trees are used for testing, on which the user model achieves a correctness rate of 99.98%.

In summary, our contributions are threefold:

- We propose a structured, extensible, and easy-to-understand textual representation for fault trees.
- We construct a benchmark for evaluating fault localization and error recovery capabilities of LLMs in multi-turn dialogue settings.
- We train a user model that generates vague, uncertain, and biased responses to support realistic multi-turn interaction evaluation.

2 Related Work

2.1 Fault Tree Analysis

Fault Tree Analysis (FTA) was originally introduced by Bell Laboratories in 1962 as a qualitative design aid for system reliability analysis(Ruijters

and Stoelinga, 2015). It was later adopted and formalized by Boeing in the aerospace industry(Eckberg, 1963), where it evolved into a practical analytical and computational tool for diagnosing and maintaining complex engineered systems(Ericson and Ll, 1999). Since then, FTA has been widely applied across safety-critical domains such as aviation(Goldberg et al., 1994), chemical process(of Chemical Engineers. Center for Chemical Process Safety, 1985), and so on. Classical FTA represents causal relationships between failures using Boolean logic, primarily through AND and OR gates, and decomposes a predefined system-level top event into basic events corresponding to component-level failures(Vesely et al., 1981). Dynamic Fault Trees(Čepin and Mavko, 2002) introduce additional logic gates and temporal dependencies, and the International Electrotechnical Commission established the IEC 61025 standard(Committee et al., 2006), which defines FTA as a globally accepted formalism for reliability and safety analysis.

2.2 Evaluation of Multi-Turn Dialogue Systems

Existing research commonly categorizes dialogue systems into task-oriented dialogue (TOD) systems and open-domain dialogue (ODD) systems(Wang et al., 2025). Our work can be viewed as an extension of TOD evaluation into the domain of fault tree–based diagnosis, which exhibits inherently multi-turn, highly structured, and domain-specific characteristics. For TOD systems, a number of widely adopted benchmarks have been proposed, including general-purpose datasets such as MultiWOZ(Budzianowski et al., 2020) and CrossWOZ(Zhu et al., 2020), as well as domain-specific benchmarks such as KddRES(Wang et al., 2020), Transfer-TOD(Zhang et al., 2024), and PFDial(Zhang et al., 2025). Compared with existing multi-turn evaluation efforts, such as MT-Eval(Kwan et al., 2024), which evaluates dialogue performance across four domains but with relatively short interaction lengths, or MT-Bench-101(Bai et al., 2024), which further introduces a hierarchical evaluation framework for multi-turn dialogue, our benchmark emphasizes long-horizon interactions involving complex reasoning over structured fault trees, and additionally incorporates a trained user simulator to generate realistic and imperfect user behaviors.

2.3 Evaluation of Agent Capabilities

Recent studies on LLM-based agents have proposed a variety of benchmarks that emphasize multi-step reasoning, tool use, memory, and self-reflection capabilities(Yehudai et al., 2025). For example, τ-Bench(Yao et al., 2024) evaluates an agent's ability to simulate customer service interactions in aviation and retail domains, while CRMArena(Huang et al., 2025) assesses agent performance in customer relationship management scenarios that require executing multi-step operations under domain-specific policies. MINT(Wang et al., 2023) evaluates planning capabilities in interactive environments, focusing on how agents adapt their strategies based on environmental feedback. In many agent benchmarks, tools are treated as interfaces with parameterized actions in uncertain environments(Patil et al., 2025). In contrast, fault trees define a set of valid structural states in advance, and the corresponding remediation actions can be determined once the underlying failures are correctly identified. Consequently, our work places less emphasis on tool-use ability.

3 The JFTA Representation

Existing fault tree representations are predominantly diagrammatic or semi-formal, typically distributed as static images(Vesely et al., 1981). While effective for human interpretation, such formats are less suited for direct processing by large language models(Tang et al., 2025). To bridge this gap, we propose a structured textual syntax based on JSON, termed JFTA (JSON-based Fault Tree Analysis). JFTA preserves the core deductive logic of FTA while providing a formal, parseable, and extensible format, and strikes a balance between natural language expressiveness and formal structural rigor. On one hand, node names and solution descriptions are presented in natural language, facilitating semantic understanding. On the other hand, the underlying architecture enforces strict syntactic and logical constraints, supporting programmatic verification. We provide detailed information and example of JFTA in Appendix A.

3.1 Syntax Definition

In JFTA, a fault tree is defined as a rooted structure that can represent a Directed Acyclic Graph (DAG). Each node constitutes a semantic unit relevant to the fault, representing a different-level event, or a logic gate. JFTA supports a flexible set of logical

and semantic nodes.

Logic Gate Nodes: These nodes formally describe the causal relationships between parent and child events. JFTA incorporates standard Boolean operators, including AND, OR, and XOR gates, to model common failure modes . The syntax is highly extensible, allowing for the integration of additional logic gates through custom definitions.

Semantic Nodes: These nodes distinguish between intermediate states that require further decomposition and terminal nodes that represent the end of the analysis. Specifically, intermediate nodes represent fault states that must be further analyzed, while leaf nodes are designated as solution nodes, containing natural language descriptions of specific repair actions or operational procedures.

3.2 Hierarchical Structure and References

The hierarchy between nodes is primarily expressed through a recursive nested structure, where a parent node contains a list of child nodes. However, complex systems often exhibit shared failure modes where a single subsystem failure contributes to multiple higher-level events. To avoid redundancy and explosion in tree size, JFTA introduces an explicit linking mechanism. Nodes can reference previously defined nodes via their unique IDs. This transforms the representation from a strict tree into a DAG, enhancing compactness and maintainability while preserving semantic consistency.

4 JFTA-Bench

To evaluate the capacity LLMs to reason over fault trees via in-context learning, we construct JFTA-Bench. Built upon the JFTA syntax defined in Section 3, this benchmark assesses a model's ability to assist users in malfunction localization within multi-turn dialogue settings. It emphasizes understanding complex context, strategic planning, and maintaining robust interaction in dynamic environments. The benchmark comprises 126 fault trees spanning diverse domains, with an average of 140 nodes per tree. From these trees, we sample 3,130 distinct fault paths, categorized by difficulty based on the number of underlying root causes. The primary metric is the model's success rate in localizing all root causes and providing correct solutions within a limited number of interaction turns. Furthermore, to simulate realistic user behavior, we explicitly introduce scenarios involving long-range state rollback and recovery. These scenarios test the model's integrated capabilities in task tracking, consistency maintenance, and error recovery when users correct previously provided information.

Figure 1 provides an overview of the JFTA-Bench construction and evaluation pipeline. The left part of Figure Figure 1 illustrates the fault tree collection process. The middle part shows how fault paths are extracted from fault trees in a bottom-up manner, capturing valid combinations of failures under complex logic gates. The right part depicts the interaction framework, where a simulated user engages with the LLM assistant under test, including scenarios with long-range state rollback and recovery .

4.1 Fault Tree Collection

We first identified 24 domains requiring the maintenance of complex systems, such as Power & Energy, Aerospace. Within these domains, we defined 126 specific failure scenarios (see Appendix B.1 for the full list).

To construct the dataset, we employed a human-in-the-loop generation pipeline. First, human experts manually annotated high-quality fault trees for three representative scenarios using strict JFTA syntax. These served as seed examples. Subsequently, we utilized GPT-4o(Hurst et al., 2024) and Claude Sonnet 4.5(Anthropic, 2025) in a One-Shot setting. By providing the complete JFTA syntax definition and the expert-annotated examples, we prompted the LLMs to generate fault trees for the remaining scenarios. The generated trees underwent rigorous review by human experts to ensure logical correctness and adherence to structural constraints, resulting in our final set of 126 verified fault trees.

4.2 Path Extraction and Dataset Construction

JFTA supports extensible logic gates and cross-branch references. Consequently, path sampling requires specific traversal logic. For instance, if a path traverses an AND gate, all child nodes of that gate must be included in the fault path. We extracted paths via a bottom-up approach, starting from sets of 1 to 6 initial basic failures. Using Algorithm 1, we extended these failures upward to the root event. Based on the number of final underlying failures derived from the expansion, the test cases are classified into three difficulty levels.

Level	Size	Assistant Prompt Length	User Prompt Length	Interaction Turns	Error Count
1	1103	5935.04	1096.87	20.17	1.50
2	1116	5924.64	1218.17	42.91	3.49
3	911	5939.75	1395.38	63.05	6.29

Table 1: Statistics of JFTA-Bench

4.3 Error Simulation and Recovery.

Real-world users often make observation errors and need to correct prior statements. To evaluate the model's robustness against such volatility, we introduced long-range state rollback and recovery scenarios. Specifically, we select two paths of identical difficulty that share a common prefix of length L_{prefix}. The interaction begins following the first path. At a random step after L_{prefix}, the user indicates that a previous observation was incorrect and switches the context to the second path. This setup tests whether the model can accurately backtrack within the dialogue history, update its internal belief state, and proceed with the correct fault diagnosis based on the revised information.

4.4 Interaction Framework

We adopt the ReAct framework (Yao et al., 2022) for the evaluation. The model under test functions as an Assistant interacting with a simulated User. In each turn, the assistant can either ask the user to verify the occurrence of a specific fault node or propose a solution for a confirmed failure. The user responds to the assistant's inquiries or confirms whether a proposed solution is effective.

To mimic the complexity of real-world diagnostics, user responses are not binary (True/False). Instead, the user provides vague or context-dependent descriptions based on the observation from environments or vague description, requiring the assistant to parse and infer the actual fault status. To achieve this, we trained a specialized User Model capable of generating ambiguous yet consistent responses, detailed in Section 5.1.

4.5 Dataset Statistics

Our dataset is derived from 126 complex fault trees, averaging 140 nodes per tree. We sampled 3130 entries, where each entry contains two distinct paths to support the error recovery evaluation. On average, each session requires 40.75 interaction turns to resolve. To support the ReAct interaction, we designed specific system prompts for both the User and Assistant (see Appendix D.2 and D.1). The objective is for the model, acting as an assistant, to start from an observed top-level symptom and progressively localize all faults and provide solutions within limited turns. Table 1 summarizes the key statistics of JFTA-Bench. The prompt length is calculated by the Qwen3-8B(Yang et al., 2025) tokenizer.

5 Experiments

5.1 User Model Training

We trained a user simulator based on Qwen3-8B to simulate realistic user behaviors, including providing ambiguous feedback and maintaining consistency. The training pipeline consists of two stages: Supervised Fine-Tuning for behavior cloning and Reinforcement Learning for response optimization.

Data Collection and Construction. We utilized the 126 fault trees collected in Section 4.1, partitioning them into a training set (100 trees) and a testing set (26 trees). For each fault tree , we constructed training samples by iterating through all possible fault nodes or solution nodes . We generated potential user responses (Accept or Reject) and sampled corresponding path sets where paths either include or exclude consistent with the response.

To improve model robustness and prevent hacking behaviors (e.g., the user leaking the entire fault path or list of failures), we augmented the dataset with negative samples. We simulated invalid assistant requests, for instance, asking for unresolved faults, querying multiple faults simultaneously, or requesting irrelevant information, and paired them with rejection responses.

We employed an ensemble of advanced LLMs, including Gemini 2.5 flash(Comanici et al., 2025b), GPT-4o, and Claude Sonnet 4.5 randomly to synthesize dialogue data via One-Shot learning. Specialized prompts were designed to encourage vague expressions, mimicking non-expert users, which is shown in Appendix D.3. The target output was formatted in JSON, containing:

`action`: Classification of the assistant's intent.
`name`: The specific fault or solution queried.
`return`: The logical feedback (Accept/Reject).
`response`: The natural language response, specifically stylized to be ambiguous.

Table 2 presents detailed statistics of the collected user simulation dataset.

Metric	Training Set	Testing Set
Total Samples	37,115	9,714
Category		
Solution	13,524	3,566
Fault	22,808	6,061
Hacking	783	87
Length		
Input	1,296.09	1,301.37
Output	51.18	51.68

Table 2: Statistics of the Training and Testing Datasets

Behavior Cloning. We first fine-tuned the Qwen3-8B model using the LLaMA-Factory(Zheng et al., 2024) framework on the collected dataset to obtain the behavior-cloned model, U_{bc} . Training was conducted on 16 NVIDIA H20 GPUs for 4 hours. Detailed hyperparameters are provided in the Appendix C.1.

Response Optimization. Inspired by recent work in rule-based reinforcement learning(Guo et al., 2025), we further optimized U_{bc} using the Proximal Policy Optimization (PPO) algorithm(Schulman et al., 2017) to improve adherence to the requirements of vagueness and the format constraints. We defined a hybrid reward function as follows:

- r_{format}: A positive reward is granted if the model output strictly follows the required JSON schema.
- $r_{correctness}$: If the format is valid, we compare the generated fields (`action`, `name`, `return`) with the ground truth .

We utilized the veRL framework(Sheng et al., 2024) for PPO training. The training was conducted on a cluster of 8 H20 GPUs. The PPO optimization process required approximately 8 hours to complete. Specific RL hyperparameters are listed in the Appendix C.2.

5.2 User Model Evaluation

To investigate the performance differences between reinforcement learning strategies, we additionally implemented the Group Relative Policy Optimization (GRPO) algorithm(Shao et al., 2024) with a group size of $N = 4$.

We evaluate the trained User Model from both functional correctness and behavioral realism perspectives. Correctness is measured in a strict, deterministic manner: a generated response is considered correct only when its `action`, `name`, and `return` fields exactly match the ground-truth annotation.

Beyond correctness, we assess the realism of the generated user responses in terms of information quality. Specifically, we adopt an LLM-as-a-Judge paradigm to evaluate whether the `response` exhibits user-like ambiguity and natural expression. The judge model scores each response along two complementary dimensions: *fuzziness compliance*, which reflects whether the description aligns with the indirect, non-diagnostic observation style of non-expert users, and *naturalness*, which measures the degree to which the expression resembles spontaneous, colloquial human speech rather than templated or system-like outputs. We employ GPT-4o as the judge model. Scores are assigned on a 0–3 scale for fuzziness and 0-2 for naturalness based on expert-designed rubrics and reference examples, ensuring consistent and principled evaluation across responses. Detailed prompt is shown in Appendix D.4.

As shown in Table 4, the base model performs poorly in both fuzziness and naturalness while All fine-tuned models achieve substantial improvements. SFT already produces more realistic user responses but tends to remain slightly rigid. PPO yields the best overall performance, whereas GRPO, despite comparable correctness, shows a mild regression in behavioral realism. Notably, although our reward design explicitly focuses only on format compliance and factual correctness, both fuzziness and naturalness improve after reinforcement learning. We hypothesize that reinforcement learning reshapes the response search space, mitigating the rigid response patterns induced by SFT and enabling more flexible, human-like expression.

5.3 Main Evaluation

We benchmarked a selection of frontier Large Language Models on JFTA-Bench. To optimize com-

Model	Error Action(%)				Correct(%)				Avg Turns
	Path	Plan	Solution	Graph	Level 1	Level 2	Level 3	Total	
DeepSeek-V3.2	11.62	57.94	14.75	15.70	55.74	37.90	30.95	41.40	52.04
Qwen3-32b	11.82	36.50	44.21	7.47	34.43	28.23	24.60	29.03	43.17
Claude Sonnet 4.5	14.64	60.49	14.18	10.69	65.57	50.00	**43.65**	52.96	50.85
Gemini 2.5 pro	9.95	48.57	28.19	13.30	**70.49**	50.00	41.27	**53.76**	52.10
GPT-5	15.02	54.13	17.74	13.11	56.56	**50.81**	**43.65**	50.27	50.03

Table 3: Performance Summary of Different Models on JFTA-Bench

Model	Fuzziness	Naturalness	Correctness
Base	0.63	0.94	91.97
SFT	2.24	1.51	99.94
PPO	**2.29**	**1.53**	**99.98**
GRPO	2.21	1.46	99.47

Table 4: Evaluation of user response quality. Base model refers to Qwen3-8B

putational efficiency, initial experiments were conducted on a representative subset of the data,and the analysis of subset consistency is provided in Section 5.5. We evaluate a diverse set of state-of-the-art LLMs, comprising proprietary models, including Gemini 2.5 pro, GPT-5, and Claude Sonnet 4.5, alongside leading open-source counterparts including Qwen3-32B(Yang et al., 2025) and DeepSeek-V3.2.

Table 3 summarizes the performance of different models in the JFTA-Bench. Among all evaluated models, Gemini 2.5 pro achieves the best overall performance, obtaining the highest correctness rate. In general, closed-source models consistently outperform open-source models across. Within the open-source group, DeepSeek-V3.2 demonstrates the strongest performance, achieving the highest correctness and more balanced error distribution.

5.4 Analyze

As shown in Table 3, the primary failure mode of the assistant lies in planning errors(counted at a node level). Specifically, the assistant may repeatedly ask about a node for which the user has already provided diagnostic feedback. Notably, the think field of model's response often indicates that the model is already capable of determining whether a fault exists, yet it still fails to advance the dialogue. Furthermore, in almost all instances where a path error occurs, the model fails to recognize the mistake and make a correction. In addition, model shows insufficient discrimination between user feedback provided before and after the rollback, leading to unsuccessful error recovery. These behaviors suggest that the model may not sufficiently exploit dialogue-based memory when making decisions, potentially due to limitations in contextual attention over long interaction histories.

A second category of failures involves path deviation, including but not limited to prematurely exiting a subtree before completing its diagnosis, fails to accurately identify the specific node that the user intends to revise within the dialogue history during rollback, and, in a small number of cases, misinterpreting the structure of the fault tree itself. These issues are likely caused by the model's limited understanding of highly complex fault tree structures, insufficient grasp of DFS traversal logic, and occasional misjudgments when reasoning over different logical gates.

Additionally, some failures arise at the solution stage, where the assistant either provides an incorrect resolution for the remaining fault or outputs a solution associated with an incorrect identifier.

5.5 Ablation Studies

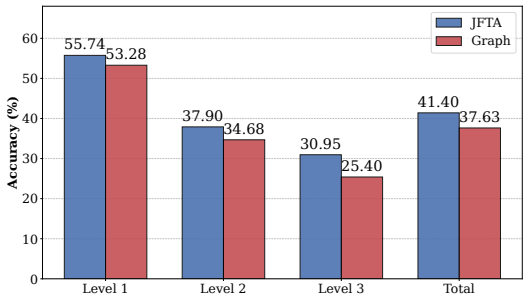

Figure 2: Accuracy comparison between JFTA and Node-Edge representations across different difficulty levels and overall performance.

Effectiveness of JFTA Representation. To demonstrate the superiority of our proposed syntax, we compared JFTA against a standard baseline representation that encodes the fault tree structure

using a list of nodes and edges (Graph format), commonly used in graph-to-text tasks. We evaluated the performance of DeepSeek-V3.2 on JFTA-Bench using both formats. The Node-Edge representation leads to substantially longer prompts due to its verbosity, with an average length of 14,669.33 tokens under the Qwen3-8B tokenizer, compared to only 5,932.7 tokens for JFTA. As illustrated in Figure 2, across all difficulty levels, JFTA consistently achieves higher accuracy than the Node-Edge representation, and the performance gap further widens as task difficulty increases. This substantial reduction in token length allows more of the effective context window to be devoted to semantically meaningful structural information rather than redundant relational descriptors, thereby improving the model's ability to preserve long-range dependencies and perform multi-step reasoning. Overall, these results indicate that the nested, hierarchical structure of JFTA is better aligned with the inductive biases of LLMs than a flat edge-list representation.

Dataset	Average Turns	Correct (%)
Subset 1	52.04	41.40
Subset 2	53.15	41.13
Subset 3	51.76	41.94
Full Dataset	**54.85**	**40.54**

Table 5: Performance comparison across different subsets and the full dataset.

Subset Consistency Analysis. Conducting full-scale evaluations on the entire JFTA-Bench dataset incurs substantial computational overhead. To mitigate this without compromising evaluation reliability, we constructed a representative test subset. Specifically, we randomly sampled entries from each level for all fault trees to form a subset containing 372 test entries. To verify the statistical consistency of this approach, we compared the performance of DeepSeek-V3.2 across the full dataset and three independently sampled subsets. The results, presented in Table 5, demonstrate that the absolute performance variance between different subsets remains less than 0.54%, and the deviation between any subset and the full dataset is less than 1.40%. This confirms that our subset evaluation strategy provides a robust and reliable approximation of the model's overall capability.

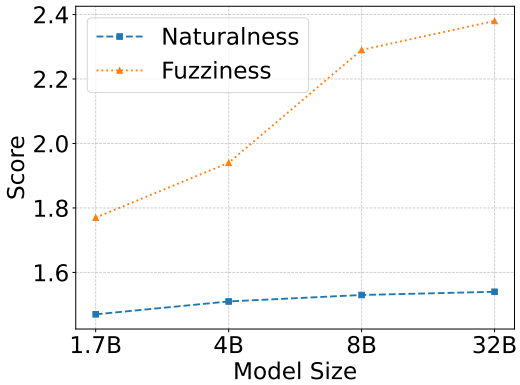

Figure 3: Performance of naturalness and fuzziness across different model sizes.

Impact of User Model Size. We investigated the impact of model size scaling on the User Simulator's performance by comparing base models of varying sizes. We ultimately selected Qwen3-8B as the backbone for our user simulator. As illustrated in Figure 3, after Behavior Cloning and Response Optimization, the 8B model significantly outperforms smaller variants in terms of response information quality. Conversely, while larger models offer marginal performance gains, they incur substantially higher training and inference costs. The 8B parameter count proves to be the optimal balance, delivering robust simulation capabilities with high computational efficiency.

6 Conclusion

In this work, we bridge the gap between visual fault tree analysis and large language models by introducing JFTA, a structured textual representation that supports rigorous logical reasoning. Building on this syntax, we construct JFTA-Bench, a comprehensive multi-turn dialogue benchmark that evaluates model performance in complex diagnostic scenarios, featuring a novel evaluation of error recovery capabilities via long-range state rollback, and a user simulator trained for generating diverse and vague responses. The benchmark highlights the remaining challenges for LLMs in maintaining consistency during dynamic, error-prone interactions and planning ability. This work lays a solid foundation for future research on LLMs in real-world applications, while also providing new insights into the analysis and practical use of LLMs in JFTA-based malfunction diagnosis and reasoning.

Limitations

Although the proposed JFTA and its benchmark perform well in evaluations, several limitations remain. First, due to strict confidentiality protocols in the industrial sector, our benchmark lacks large-scale, native fault tree data collected directly from real-world production lines. Consequently, the current semi-synthetic dataset may not fully capture the unpredictable noise and complexity of rare industrial malfunctions, which restricts a comprehensive evaluation of a model's diagnostic reasoning capabilities in real-world scenarios. Second, the long-horizon, multi-turn interactions required for evaluation incur substantial computational overhead, necessitating the use of smaller representative subsets for certain analyses. Finally, JFTA has not yet been deployed within operational industrial maintenance software; its practical integration challenges, real-time performance, and usability in live production environments remain to be validated.

References

Anthropic. 2025. Claude 4.5 sonnet system card. Technical report, Anthropic PBC. Technical Report.

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and 1 others. 2024. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7421–7454.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2020. Multiwoz – a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *Preprint*, arXiv:1810.00278.

Marko Čepin and Borut Mavko. 2002. A dynamic fault tree. *Reliability Engineering & System Safety*, 75(1):83–91.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 3416 others. 2025a. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025b. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

IEC 61025 Technical Committee and 1 others. 2006. Fault tree analysis (fta). *IEC 61025*.

DeepSeek-AI. 2025. Deepseek-v3.2: Pushing the frontier of open large language models.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA. Association for Computational Linguistics.

CR Eckberg. 1963. Ws-133b fault tree analysis program plan. *Seattle, Washington: The Boeing Company*.

Clifton A Ericson and Clifton Ll. 1999. Fault tree analysis. In *System Safety Conference, Orlando, Florida*, volume 1, pages 1–9.

Ze Feng, Jiang jiang Liu, Sen Yang, Lingyu Xiao, Zhibin Quan, Zhenhua Feng, Wankou Yang, and Jingdong Wang. 2025. Vision remember: Recovering visual information in efficient lvlm with vision feature resampling. *Preprint*, arXiv:2506.03928.

Gemini Team, Google. 2025. Gemini 3 pro model card.

Benjamin E Goldberg, K Everhart, R Stevens, N Babbitt III, P Clemens, and L Stout. 1994. System engineering toolbox for design-oriented engineers. Technical report.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. 2025. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3830–3850.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Durga Rao Karanki, V.G Gopika, V. Rao, H.s Kushwaha, A.K. Verma, and A. Srividya. 2009. Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment. *Reliability Engineering & System Safety*, 94:872–883.

Wai-Chung Kwan, Xingshan Zeng, Yuxin Jiang, Yufei Wang, Liangyou Li, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. 2024. Mt-eval: A multi-turn capabilities evaluation benchmark for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20153–20177.

Wen-Shing Lee, Doris L Grosh, Frank A Tillman, and Chang H Lie. 2009. Fault tree analysis, methods, and applications a review. *IEEE transactions on reliability*, 34(3):194–203.

Fangyu Liu, Julian Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. 2023a. DePlot: One-shot visual language reasoning by plot-to-table translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10381–10399, Toronto, Canada. Association for Computational Linguistics.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023b. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, and 1 others. 2025. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*.

American Institute of Chemical Engineers. Center for Chemical Process Safety. 1985. *Guidelines for Hazard Evaluation Procedures*. American Institute of Chemical Engineers.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Enno Ruijters and Mariëlle Stoelinga. 2015. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 15:29–62.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Aaditya K. Singh, Adam Fry, Adam Perelman, Adam Tart, Adithya Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr A Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 464 others. 2025. Openai gpt-5 system card.

Jianting Tang, Yubo Wang, Haoyu Cao, and Linli Xu. 2025. Basic: Boosting visual alignment with intrinsic refined embeddings in multimodal large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20582–20592.

William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. 1981. Fault tree handbook. Technical report.

Hongru Wang, Min Li, Zimo Zhou, Gabriel Pui Cheong Fung, and Kam-Fai Wong. 2020. Kddres: A multi-level knowledge-driven dialogue dataset for restaurant towards customized dialogue system. *arXiv preprint arXiv:2011.08772*.

Hongru Wang, Lingzhi Wang, Yiming Du, Liang Chen, Jingyan Zhou, Yufei Wang, and Kam-Fai Wong. 2025. A survey of the evolution of language model-based dialogue systems: Data, task and models. *Preprint*, arXiv:2311.16789.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*.

Ming Zhang, Caishuang Huang, Yilong Wu, Shichun Liu, Huiyuan Zheng, Yurui Dong, Yujiong Shen, Shihan Dou, Jun Zhao, Junjie Ye, and 1 others. 2024. Transfertod: A generalizable chinese multi-domain task-oriented dialogue system with transfer capabilities. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12750–12771.

Ming Zhang, Yuhui Wang, Yujiong Shen, Tingyi Yang, Changhao Jiang, Yilong Wu, Shihan Dou, Qinhao Chen, Zhiheng Xi, Zhihao Zhang, Yi Dong, Zhen Wang, Zhihui Fei, Mingyang Wan, Tao Liang, Guojun Ma, Qi Zhang, Tao Gui, and Xuanjing Huang. 2025. PFDial: A structured dialogue instruction fine-tuning method based on UML flowcharts. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2626–2649, Vienna, Austria. Association for Computational Linguistics.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2026. A survey of large language models. *Preprint*, arXiv:2303.18223.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

Qi Zhu, Kaili Huang, Zheng Zhang, Xiaoyan Zhu, and Minlie Huang. 2020. Crosswoz: A large-scale chinese cross-domain task-oriented dialogue dataset. *Transactions of the Association for Computational Linguistics*, 8:281–295.

A JFTA Details

A.1 Structural Constraints

To ensure the validity of the representation for automated reasoning, JFTA imposes graph-theoretic constraints. The overall structure must remain acyclic to prevent infinite reasoning loops, and solution nodes must exclusively appear at the leaf level. These constraints align JFTA with the fundamental assumptions of classical FTA while facilitating efficient parsing and consistency checking.

A.2 Implementation detail

In the implementation, each node is uniquely identified by an ID that serves as the key in the JSON representation, while the remaining attributes are organized as the corresponding value in a dictionary structure. Logical gate behavior is thus embedded within semantic nodes through their associated type definitions. Table 6 summarizes the fields of a basic JFTA node.

A.3 Fault Tree Examples

Fault Tree Examples

```
{
 "1": {
  "NodeName": "Light does not turn on
    ",
  "NextType": "XOR",
  "NextTree": {
   "2": {
    "NodeName": "Switch Closed",
    "NextType": "OR",
    "NextTree": {
     "4": {
      "NodeName": "Power Supply Issue
    ",
      "NextType": "OR",
      "NextTree": {
       "8": {
        "NodeName": "Power not
    connected",
        "NextType": "Solution",
        "NextTree": "Connect power"
       },
       "9": {
        "NodeName": "Power plug loose
    ",
        "NextType": "Solution",
        "NextTree": "Secure power
    plug"
       },
       "10": {
        "NodeName": "Insufficient
    voltage",
        "NextType": "Solution",
        "NextTree": "Install
    transformer"
       }
      }
     },
     "5": {
      "NodeName": "Circuit Issue",
      "NextType": "Fault",
      "NextTree": {
       "13": {
        "NodeName": "Circuit Issue (
    Test intermediate faults)",
        "NextType": "OR",
        "NextTree": {
         "11": {
          "NodeName": "Wire breakage
    ",
          "NextType": "Solution",
          "NextTree": "Replace wire"
         },
         "12": {
          "NodeName": "Poor contact",
          "NextType": "Solution",
          "NextTree": "Repair contact
     point"
         }
        }
       }
      }
     },
     "6": {
      "NodeName": "Bulb Issue",
      "NextType": "Solution",
      "NextTree": "Replace bulb"
     },
     "7": {
      "NodeName": "Switch Broken",
      "NextType": "Solution",
      "NextTree": "Replace switch"
     },
     "14": {
      "NodeName": "Power Supply Issue
     (Test DAG)",
      "NextType": "LINK",
      "NextTree": "4"
     },
     "15": {
      "NodeName": "Voltage Issue (
    Test DAG)",
      "NextType": "LINK",
      "NextTree": "10"
     }
    }
   },
   "3": {
    "NodeName": "Switch Open",
    "NextType": "Solution",
    "NextTree": "Close switch"
   }
  }
 }
}
```

Field Name	Type	Description
ID	String	A globally unique identifier within the fault tree.
NodeName	String	Natural language text describing a fault phenomenon, cause, or solution.
NextType	Enum	Specifies the logical type and behavior of the node, including `XOR`, `OR`, `AND`, `Fault`, `Solution`, and `LINK`. This enumeration is extensible to support additional types.
NextTree	Object	When `NextType` is `XOR`, `OR`, `AND`, or `Fault`, this field is an object containing child nodes.
	String	When `NextType` is `Solution`, this field contains a textual description of the solution; when it is `LINK`, this field stores the referenced node ID.

Table 6: Fields of a JFTA Basic Node

B Dataset Details

B.1 Domains and scenarios

The table 7 and table 8 present the domains and failure scenarios included in the data.

B.2 Path Sampling Algorithm

Algorithm 1 presents the algorithm for sampling paths by expanding siblings.

Algorithm 1: Path Sampling Algorithm

Input: Fault Tree G, Root r, Basic Solutions S_{init}
Output: Selected Path Nodes $V_{selected}$

Function PATHSAMPLING(G, r, S_{init})
Initialize $V_{selected} \leftarrow \emptyset$
for all $s \in S_{init}$ **do**
  TRACEBACKTOROOT(s, r)
end for
repeat
  $S_{new} \leftarrow$ EXPAND($V_{selected}$)
  if $S_{new} \neq \emptyset$ **then**
    for all $s \in S_{new}$ **do**
      TRACEBACKTOROOT(s, r)
    end for
  end if
until S_{new} is empty
return DFS($V_{selected}$)

Function TRACEBACKTOROOT(u, r)
if $u \in V_{selected}$ **then**
  return
end if
$V_{selected} \leftarrow V_{selected} \cup \{u\}$
if $u \neq r$ **and** $Parents(u) \neq \emptyset$ **then**
  $p \leftarrow$ Randomly select one from $Parents(u)$
  TRACEBACKTOROOT(p, r)
end if

Function EXPANDANDSIBLINGS(V_{curr})
$S_{added} \leftarrow \emptyset$
for all $u \in V_{curr}$ **do**
  if NEEDEXPEND(u) **then**
    for all child $c \in Children(u)$ **do**
      if $c \notin V_{curr}$ **then**
        $S_{sub} \leftarrow$ GETONESOLUTION(c)
        $S_{added} \leftarrow S_{added} \cup S_{sub}$
      end if
    end for
  end if
end for
return S_{added}

Domain	Failure Scenarios
Power & Energy	Large-scale power grid blackout; Nuclear power plant safety system failure; Petrochemical plant explosion/leakage; Wind power/Photovoltaic (PV) system fault; Energy storage station fire/explosion; Smart grid data tampering.
Aerospace	Aircraft engine failure; Flight control system paralysis; Spacecraft launch failure; Satellite in-orbit service failure; UAV (Drone) crash/loss of contact.
Automotive & Transportation	Automotive braking system failure; Railway signaling system fault; Subway system suspension; Ship propulsion/navigation fault; Electric Vehicle (EV) thermal management failure; High-level autonomous driving misjudgment.
Medical & Life Sciences	Ventilator/Pacemaker failure; Hospital power/oxygen supply interruption; Medical robot malfunction; In-vitro diagnostics (IVD) equipment misdiagnosis; Biobank temperature control failure.
Information & Communication	Data center downtime; Communication network interruption; Satellite communication failure; Large-scale cloud service outage; Content Delivery Network (CDN) scheduling anomaly.
Construction & Public Safety	Fire protection system failure; Elevator suspension accident; High-rise building water supply interruption; Smart building security system failure; Large stadium evacuation failure; Supercomputer center liquid cooling leak.
Mining & Industrial Safety	Gas explosion accident; Mine flooding/roof collapse; Chemical production unit failure; Industrial robot injury accident; Polymerization reactor runaway; Hazardous chemical storage chain explosion; Digital factory industrial control network paralysis.
Defense & Military	Weapon system malfunction; Military vehicle/vessel failure; Battlefield communication interruption; Reconnaissance satellite data transmission failure.
Microelectronics & Semiconductors	Chip tape-out failure; Wafer manufacturing low yield; Advanced packaging reliability failure; Integrated circuit electrostatic discharge (ESD) breakdown.
Computer Systems & Software	Operating system kernel panic; Distributed database data inconsistency; Online payment system double deduction; AI model performance degradation in production; Recommender system filter bubble reinforcement; Smart home system failure; Cloud computing resource scheduling imbalance.
Emerging Tech & Frontiers	Quantum computer fidelity drop; Gene editing off-target effects; Brain-Computer Interface (BCI) command misinterpretation; Metaverse virtual space motion sickness; Synthetic biology engineered strain escape; Digital twin vs. physical entity distortion.

Table 7: Potential Failures in Various Domains (Part 1)

Domain	Failure Scenarios
Finance & Infrastructure	Stock exchange trading system interruption; Automated Market Maker (AMM) protocol manipulation; Digital currency wallet private key loss; AI financial risk control model failure.
Environment & Ecology	Chemical plant toxic substance leak into river; Nuclear waste geological repository leakage; Carbon Capture and Storage (CCS) failure; Deep-sea aquaculture cage mass fish mortality; Chemical wastewater treatment system failure.
Public Health & Biosecurity	Pathogen laboratory leak; Public health surveillance system reporting failure; Uncontrolled community outbreak of infectious disease; Mass vaccination failure.
Agriculture & Food Engineering	Smart irrigation system failure; Precision fertilization system disorder; Greenhouse environment control loss; Livestock barn environment deterioration; Cold chain logistics break; Transgenic crop gene drift.
Ocean Engineering & Polar	Deep-sea drilling platform blowout; Wave energy device structural fatigue fracture; Polar station energy supply chain break; Autonomous Underwater Vehicle (AUV) mission failure; Submarine optical cable communication interruption; Icebreaker propulsion system disability.
Sports Events	Smart referee system misjudgment; Stadium energy system failure; Event broadcast signal interruption; Ticketing system crash; Athlete monitoring equipment failure.
Tourism Attractions	Scenic area passenger flow monitoring failure; Amusement ride suspension accident; Smart guide platform downtime; Artifact storage constant temp/humidity system anomaly; Emergency evacuation system failure.
Education	Online teaching platform downtime; Smart exam system cheating/failure; Campus "One Card" system anomaly; Teaching building power outage; Virtual laboratory simulation error.
Urban Infrastructure	City-wide traffic light paralysis; Urban heating system failure; Smart streetlight system anomaly; Underground drainage pump station failure; City emergency broadcast malfunction; Smart parking system crash.
Media & Entertainment	Large-scale event safety accident; Live streaming platform massive downtime; Sports event broadcast interruption; News gathering/editing system data loss; Short video recommendation anomaly; Digital Rights Management (DRM) crash; False news reporting.
Judiciary	Electronic court system downtime; Electronic evidence chain failure; Prison security system failure; Notary office electronic signature system downtime; Judicial appraisal laboratory equipment fault.
Rail Transit	Signaling system fault; Train traction system failure; Train braking system failure; Platform screen door system anomaly; Train Control System (CBTC/ATO) crash.

Table 8: Potential Failures in Various Domains (Part 2)

C Hyperparameters

C.1 Supervised Finetuning

The table 9 presents the hyper-parameters used for Supervised Fine-Tuning.

Hyperparameter	Value
Batch Size	64
Learning Rate	1.0×10^{-5}
LR Scheduler	Cosine
Optimizer	AdamW
Warmup Ratio	0.1
Epochs	3
Precision	bf16

Table 9: Hyperparameters used for SFT.

C.2 Reinforcement Learning

The table 10 presents the hyper-parameters used for PPO Training.

Hyperparameter	Value
Total Epochs	3
Advantage Estimator	GAE
Global Train Batch Size	256
Precision	bf16
Actor Learning Rate	1.0×10^{-6}
PPO Mini-Batch Size	64
KL Loss Coefficient	0.001
Optimizer	AdamW
Critic Learning Rate	1.0×10^{-5}
Critic Warmup	0
Max Response Length	1024

Table 10: Hyperparameters used for PPO Reinforcement Learning.

D Prompts

D.1 Prompt for Assistant

Prompt for Assistant

```
You are now acting as a technical
    support expert in the **{field}**
     domain. Your task is to assist
    users in systematically
    troubleshooting faults and
    providing final solutions.

---

## Task Objective

The fault tree consists of three
    levels: top-level, intermediate-
    level, and bottom-level:

* Bottom-level faults are the root
    causes of the problem and can be
    directly fixed;
* Intermediate-level and top-level
    faults are higher-level
    manifestations of bottom-level
    faults.

The user will initially describe a
    top-level fault. Your task is to:

1. Start from the top-level node and
    conduct step-by-step
    troubleshooting down the fault
    tree;
2. Gradually identify the true root
    cause by confirming with the user
     whether a specific intermediate
    or bottom-level fault exists;
3. Once a bottom-level fault is
    confirmed, provide a clear,
    actionable, and engineering-
    feasible solution;
4. A single scenario may involve
    multiple bottom-level faults;
    only after all of them are
    resolved can the issue be
    considered fully fixed.

You will also receive the user's
    historical troubleshooting
    records and solution verification
     feedback to maintain consistency
     and continuity of the fault tree
     state.

Below is the fault tree structure you
     have access to:
{fault_tree}

---

## Output Format

Always respond using the following
    JSON format:

```json
{
    "think": "Based on the user's
    response...",
    "ID": "string",
    "response": "The troubleshooting
    action to perform in this round
    or the solution to be provided"
}
```

Response Rules

1. Node ID Rules
```

```
The "ID" field must correspond to the
     **node ID actually being handled
     in the next turn**:

* If this round involves
    troubleshooting, fill in the ID
    of the node you are querying;
* If this round provides a solution,
    fill in the ID of the
    corresponding bottom-level fault.

It must be consistent with the
    operation in the next dialogue
    turn.

---

### 2. Assistant Response Rules

When troubleshooting or providing
    solutions, responses must be
    naturally phrased and maintain a
    professional technical tone,
    while explicitly stating the **
    full fault name** currently being
     handled. Responses should be
    concise (within 20 words), only
    describing the *object to be
    checked* or the *action to be
    taken*, without elaborating on
    details.

To avoid templated outputs,
    expressions must remain varied.
    Do not use the same sentence
    structure in two consecutive
    turns, and do not mechanically
    repeat fixed patterns such as "
    need to confirm whether ... fault
     exists."

**During the troubleshooting phase,
    the response must include the
    full fault name.**
**During the solution phase, the
    response must include both the
    full fault name and the method.**

> **Mandatory requirements**:
>
> * Do not repeat the same sentence
    structure within three turns.
> * Do not use fixed templates such
    as "XXX is usually handled by
    repair or replacement."
> * Do not use templated phrasing
    like "need to confirm whether XXX
     fault exists."

---

### 3. Think Field

This field is used to explain your
    reasoning process, including but
    not limited to:

* How you ruled out or confirmed a
    node based on the user's previous
     feedback;
* Why the current node should be
    handled in this round;
* Why it is appropriate to enter the
    solution phase at this point.

The thinking content does not need to
     repeat node IDs or solution
    details; it should only reflect
    the reasoning process.

---

## Examples

### Troubleshooting Example

```json
{
    "think": "The user has ruled out
    transmission-side anomalies, so
    the investigation should shift to
     the generation branch",
    "ID": "2",
    "response": "Next, the generator
    operation fault needs to be
    examined"
}
```

Solution Example

```json
{
    "think": "The user has confirmed
    material fatigue, which is a
    bottom-level fault and can be
    directly addressed",
    "ID": "23",
    "response": "Material fatigue can
     be resolved by replacing the
    impeller assembly"
}
```
```

D.2 Prompt for User

Prompt for User

```
You are now required to role-play as
    a user working in the **{field}**
     domain, collaboratively
    troubleshooting an occurring
    fault with the help of an
    assistant.

Your task is: after receiving a
    message from the assistant, first
     determine whether the assistant
    is currently performing **"Fault
    Confirmation"** or **"Solution
    Confirmation"**, and then return
    the corresponding JSON according
    to the reply rules.
```

926

Known Fault Information

* **Fault Path (from top level to bottom level)**:
  {error_paths}

* **Bottom-level Faults and Their Corresponding Solutions**:
  {solutions}

Output Format

Your output must **strictly contain only one JSON object**, in the following format, and **no other content should be included**:

```json
{
  "action": "",
  "name": "",
  "return": "",
  "response": ""
}
```

Reply Rules

You must first determine the assistant's action type based on its message, then fill in the remaining fields according to the rules for that action. The assistant has two valid action types: **"Fault Confirmation"** and **"Solution Confirmation"**. If the message does not fall into either category, it should be treated as **"Invalid Input."**

Fault Confirmation

* If the assistant is asking whether a specific fault exists or requesting you to check the status of a certain node, it is considered **"Fault Confirmation."**

* When the assistant is performing **"Fault Confirmation,"** you must further determine whether the fault exists. Specifically:

  * If the fault name and ID provided by the assistant exactly match a node in the known fault path, set the `"return"` field to `"True"`, indicating that the fault exists.
  * Otherwise, set the `"return"` field to `"False"`, indicating that the fault does not exist.
  * Accurately fill the `"name"` field with the node name that the assistant wants to investigate.
  * The `"response"` field should contain a natural-language reply to the assistant. You must convert your judgment into **vague and non-committal user feedback typical of real production environments**, following these rules:

    * **Do not directly state that a problem has occurred or has not occurred**, and do not use definitive or conclusive expressions.
    * **Components, devices, or modules** involved in the fault should be referred to indirectly via their function, purpose, or appearance.
    * **States, faults, or anomalies** should be replaced with vague sensations, suspected phenomena, observed changes, or possible impacts, without identifying causes, drawing conclusions, or making declarative statements.
    * The rewritten sentence should be natural, conversational, and consistent with the ambiguous expressions commonly used by on-site personnel, while preserving semantic correspondence with the original meaning.

* Examples:

```json
{
  "action": "Fault Confirmation",
  "name": "Chip Overheating",
  "return": "True",
  "response": "I've noticed that the core area responsible for signal processing seems to be giving off more heat than before."
}
```

```json
{
  "action": "Fault Confirmation",
  "name": "Insulation Aging",
  "return": "False",
  "response": "The outer layer used to isolate the circuitry still looks fairly normal for now, and I haven't noticed anything unusual."
}
```

Solution Confirmation

* If the assistant provides operational steps or describes

927

```
    how to repair or handle the issue
    , it is considered **"Solution
    Confirmation."**

* When the assistant is performing
    **"Solution Confirmation,"** you
    must further determine whether
    the proposed method correctly
    corresponds to a bottom-level
    fault and its solution:

  * If the fault and solution both
    exist in the known fault
    information and are correctly
    matched, set `"return"` to `"True
    "`, fill the `"name"` field with
    the corresponding bottom-level
    fault name, and write a natural-
    language response in the `"
    response"` field indicating that
    the issue has been resolved.
  * Otherwise, set `"return"` to `"
    False"`, fill the `"name"` field
    with the fault name specified by
    the user, and write a natural-
    language response in the `"
    response"` field indicating that
    the solution is ineffective.

* Examples:

```json
{
  "action": "Solution Confirmation",
  "name": "Misfire",
  "return": "True",
  "response": "This method is
    effective and the issue has been
    resolved."
}
```

```json
{
  "action": "Solution Confirmation",
  "name": "Calibration Pressure
    Offset",
  "return": "False",
  "response": "The method is
    ineffective."
}
```

Invalid Input

* The following situations are
    considered **"Invalid Input"**:

  * The assistant's message cannot be
     associated with either **"Fault
    Confirmation"** or **"Solution
    Confirmation,"** such as
    indicating the investigation is
    complete or asking whether there
    are any other issues.
  * The assistant attempts to confirm
     multiple faults at the same time
    .
  * The object of solution
    confirmation is not a bottom-
    level fault.

* If the assistant's message is
    judged to be **"Invalid Input,"**
     set all fields except `"action"`
     to empty strings.

* Example:

```json
{
  "action": "Invalid Input",
  "name": "",
  "return": "",
  "response": ""
}
```
```

D.3 Prompt for User Data Collection

Prompt for User Data Collection

```
You are an expert at transforming
    clear, direct technical
    descriptions into vague,
    ambiguous, and colloquial on-site
     observational expressions. Your
    task is to take the specific
    sentences I provide and convert
    the explicit problems, components
    , or statuses within them into
    vague, non-standard descriptions.

Rewriting Rules:

* **Do not directly state "a problem
    occurred" or "no problem occurred
    "**, and do not use conclusive
    expressions.
* **Components, equipment, or parts**
     in the original text can be
    referred to indirectly by their
    purpose, function, or visual
    characteristics.
* **Statuses, faults, or anomalies**
    in the original text should be
    replaced with vague impressions,
    suspected phenomena, observed
    changes, or potential impacts,
    without diagnosing the cause,
    drawing conclusions, or
    categorizing the issue.
* The rewritten sentences must be
    natural, conversational, and
    align with the vague phrasing
    commonly used by field personnel,
     while still corresponding to the
     core meaning of the original
    sentence.

Examples:
```

```
* Original: **Encountered a chip
    overheating fault**
Obfuscated: **I noticed that the core
     area responsible for processing
    signals seems to be giving off
    more heat than before.**
* Original: **Did not encounter
    insulation aging**
Obfuscated: **I looked at that outer
    skin used to isolate the circuits
    , and it looks pretty normal
    right now; I haven't noticed any
    particular changes yet.**

You only need to output the
    obfuscated on-site phenomenon
    descriptions. No explanations or
    additional content are required.
```

D.4 Prompt for User Evaluation

Prompt for User Evaluation

```
You are an **evaluation model**
    designed to assess the output
    quality of a *fault-diagnosis
    user simulation model*.

Your input consists of two parts:

* `query`: a **fault existence check
    request** from a real production
    environment
* `response`: the **troubleshooting
    reply** generated by the user
    simulation model

Your task is to **objectively score**
     the quality of the given `
    response` across multiple
    dimensions and provide **
    verifiable justifications**.

---

## Evaluation Dimensions and Rules

### Fuzziness Compliance

**Objective**: Evaluate whether the
    description aligns with the **
    vague, non-expert observation
    style of real users**
**Maximum score: 3**

**Scoring criteria:**

* **0 points**

  * Makes an explicit conclusion

* **1 point**

  * Insufficient fuzziness
  * Contains obvious conclusive or
    judgmental language
  * But still includes a small number
     of subjective hedging words (e.g
    ., "feels like")

* **2 points**

  * Generally fuzziness-compliant
  * But some words imply judgment
  * Or clues are overly abstract (e.g
    ., "the state doesn't seem right
    ", "looks kind of weird")

* **3 points (must satisfy all of the
     following)**

  1. Uses **indirect references** (e.
    g., "seems", "feels", "kind of",
    "looks like")
  2. Provides **specific, perceptible
     clues**
  3. Contains **no state-level or
    conclusive terms**

**Penalties (-1 point per item,
    minimum 0):**

* Any of the following word types
    appear:

  * Direct conclusion terms: *exists
    / does not exist / normal / fault
     / broken*
  * State-level judgment terms: *
    stable, smooth, no issue,
    something wrong*
  * Range conclusions: *within normal
     range, not exceeding limits*
* Use of engineering terminology or
    professional diagnostic language

---

### Naturalness

**Objective**: Evaluate whether the
    expression resembles **natural
    speech from real on-site
    personnel**
**Maximum score: 2**

**Scoring criteria:**

* **0 points**

  * Clearly unnatural
  * Strong machine-like or
    engineering tone

* **1 point**

  * Semantically clear
  * But template-like or overly
    formal
  * Or sentence structure is highly
    similar to responses at other
    nodes
```

```
* **2 points**

  * Fully conversational
  * No template feel
  * Expression does not rely on fixed
     sentence patterns

**Penalties (-1 point per item,
    minimum 0):**

* Nearly universal across all nodes
* Obvious boilerplate or fixed
    phrasing

> **Strong template trigger example
    :**
> `I checked + X + currently + feels
    like + state word + temporarily
    no + abnormality`
>
> Appearance of this structure -
    naturalness less or equal than 1

---

## Output Format (Strict Requirement)

Output **only JSON**, with no
    additional explanatory text:

```json
{
  "scores": {
    "fuzziness_compliance": 0,
    "naturalness": 0
  },
  "total_score": 0,
  "comments": ""
}
```

Requirements for the `comments`
    Field (Mandatory)

* Must explicitly identify:

  * At least **one concrete issue**
  * At least **one concrete strength
    **
* Vague praise is not allowed
* Must quote **specific wording or
    structural features** from the `
    response`
* If the comments mention any of the
    following issues:

  * "lack of concrete perceptual
    details"
  * "conclusive wording"
  * "template-like expression"
  * "overly generic description"
    - the corresponding dimension **
    must be genuinely penalized**

Mandatory Consistency Constraints

1. **No concrete observable clues**
   - fuzziness_compliance cannot be 3

2. **Presence of state-level
    conclusion terms**
   - fuzziness_compliance less or
    equal than 2

3. **Template-based sentence
    structure**
   - naturalness less or equal than 1

4. **Issues mentioned in comments**
   - must be reflected in score
    deductions

Anchored Examples

Example A (High Score)

> "I just checked the charging port.
    After plugging it in, the light
    didn't turn on immediately - it
    blinked once after two or three
    seconds. Feels a bit odd."

* fuzziness_compliance: 3
* naturalness: 2

Example B (Medium Score)

> "I think the charging part seems
    fairly stable right now, and I
    didn't notice any issues."

* fuzziness_compliance: 1 ("stable",
    "didn't notice any issues" =
    state conclusions)
* naturalness: 2

Example C (Low Score)

> "The charging system is operating
    normally."

* fuzziness_compliance: 0
* naturalness: 1

Example D (Template Penalty)

> "I checked the system and it
    currently feels quite stable,
    with no abnormalities observed so
     far."

* fuzziness_compliance: 2
* naturalness: 1
```

```
---

## Notes

* All scores must strictly fall
    within the specified ranges
* Do not speculate about components
    or issues not mentioned in the `
    query`
* Do not claim problems without
    deducting points
* Do not use comments to offset
    actual score deductions

---

## Input

Please score the following sample:

{sample}

Follow the format strictly and output
     **JSON only**.
```