Structured Reasoning for Fault Tree Generation through Schema-Guided Decomposition

Jisuk Kim[a], Karen D. Souza[a], Tao Liu[a], Zhegang Ma[a], Brandon Biggs[a], Steven Prescott[a], Ruihong Huang[b], Kangda Wei[b], Vaibhav Yadav[a*]
[a] Idaho National Laboratory, Idaho Falls, ID, USA, Jisuk.kim@inl.gov
[b] Texas A&M University, College Station, TX, USA

Abstract: Safety analysis for nuclear reactors is essential to licensing and operational assurance, yet it remains resource-intensive and dependent on multidisciplinary expertise. These analyses require the integration of diverse engineering documents, structured logical modelling, and domain-specific reasoning under strict regulatory constraints. To support advanced reactor safety workflows, an agentic architecture leveraging generative artificial intelligence (AI) is under development at Idaho National Laboratory. The architecture decomposes safety analysis processes into specialized agents, each assigned a defined analytical role and operating under task-specific reasoning constraints. This multi-agentic structure enables the separation of data interpretation, structured safety analysis, and logical model construction into coordinated but distinct components.
Within this framework, this paper presents a schema-guided structured reasoning approach for fault tree generation. Rather than generating fault trees directly from system descriptions, the proposed approach constructs a safety analysis schema that captures functions, configurations, systems, components, and associated failure modes as an intermediate representation of engineering knowledge. The generated schema can be reviewed and refined by analysts through interactive interfaces that support evidence tracing and knowledge validation prior to fault tree construction. The reviewed schema is subsequently transformed into fault tree artifacts through structured logic conversion rules. To support model validation, a minimal cut set (MCS)-based evaluation methodology is introduced to compare generated and reference fault trees based on logical equivalence rather than structural similarity. The proposed approach provides a transparent and human-reviewable workflow for AI-assisted fault tree generation and validation.

1. INTRODUCTION

Fault tree analysis (FTA) is one of the most widely used methods for safety and reliability assessment in the nuclear industry and many other engineering domains [1,2]. FTA defines a system failure or loss of a safety function as a top event and systematically decomposes the causes leading to that event through a hierarchical logical structure. The resulting fault tree serves as a key input model for both qualitative safety evaluations and probabilistic risk assessment (PRA).
Traditional fault tree development requires extensive domain expertise. In conventional FTA, analysts typically adopt a top-down approach by first defining an undesired system failure or loss of a safety function as the top event [1]. They then systematically decompose the causes of that event into lower-level system failures, component failures, and applicable failure events using logical relationships until basic events are identified. This process requires a detailed understanding of system functions, configurations, dependencies, and failure mechanisms. Consequently, fault tree development is labor-intensive, time-consuming, and often requires repeated revisions when design changes occur.
Recent advances in large language models (LLMs) have demonstrated promising capabilities in technical document analysis, information extraction, and engineering knowledge generation [3]. As a result, there is growing interest in applying generative artificial intelligence to support engineering design and safety analysis activities. However, unlike conventional tasks such as document summarization or question answering, fault tree generation requires the modeling of complex engineering relationships and hierarchical dependencies, making direct application of LLMs challenging. For example, analysts must examine which systems are required to perform a given safety function, which component failures can cause system failure, and which failure modes are applicable

to each component. These relationships must be represented in a structured and traceable manner to ensure that the resulting fault tree accurately reflects the underlying engineering logic.
In addition, several studies have demonstrated the feasibility of automatically generating fault trees from structured engineering models such as architecture analysis and design language [4] and SysML [5]. However, the evaluation of generated fault trees has often focused on demonstrating generation capability through case studies, whereas systematic validation against expert-developed fault trees has received comparatively less attention.
The objective of this study is to develop a generative AI-based framework capable of automatically generating fault trees from system descriptions. However, when LLMs are used to generate fault trees directly from system descriptions, several challenges arise, including missing system or component information, lack of traceability between functions and failures, inconsistent naming of identical components, non-standardized failure mode descriptions, and difficulty validating generated fault tree logic [6,7]. Moreover, evaluating the correctness of a generated fault tree is inherently challenging because different fault tree structures may represent the same underlying logic, making direct structural comparison insufficient. These challenges indicate the need for both a structured knowledge representation framework and a systematic evaluation methodology. Therefore, generative AI-based fault tree generation should be approached as a structured engineering knowledge generation problem rather than a free-form text generation task, and the generated fault trees should be validated.
To address these challenges, this study introduces a safety analysis schema that explicitly represents the engineering knowledge required for fault tree development. Rather than generating a fault tree directly from a system description, the proposed approach first constructs a structured intermediate representation and subsequently transforms it into a fault tree model. The schema is composed of five core entities: function, configuration, system, component, and failure mode. The entities within the schema follow a hierarchical organization from functions to failure modes.
Using this representation, information extracted from system descriptions can be systematically organized before fault tree generation. The resulting intermediate representation improves traceability, consistency, and explainability throughout the generation process while providing a structured foundation for automated fault tree construction and future applications of generative AI in safety analysis and risk assessment.
In addition to schema construction, the proposed framework incorporates interactive review capabilities that enable analysts to inspect, validate, and refine generated engineering knowledge prior to fault tree generation. To support objective assessment of generated models, this study also introduces a minimal cut set (MCS)-based evaluation methodology that compares generated and reference fault trees based on logical equivalence rather than structural similarity.
Specifically, this study (1) introduces a safety analysis schema as an intermediate representation for fault tree generation, (2) provides a human-reviewable workflow for schema validation and refinement, and (3) proposes an MCS-based methodology for evaluating logical agreement between generated and reference fault trees.

2. SAFETY ANALYSIS SCHEMA-BASED FAULT TREE CONSTRUCTION FRAMEWORK

2.1. Framework Overview

This study proposes a schema-guided fault tree construction framework in which a safety analysis schema is first constructed as an intermediate representation and subsequently transformed into a fault tree model. The proposed approach is based on the observation that fault tree generation requires structured engineering knowledge rather than direct text generation. Therefore, instead of generating a fault tree directly from a system description, the framework decomposes the overall task into a series of sequential knowledge extraction, integration, and transformation processes.
As shown in Figure 1, the framework begins with the analysis of a system description to identify the engineering entities required for fault tree development. The extracted entities are then organized through relationship identification and enriched with failure knowledge obtained from predefined failure taxonomies and historical safety analysis information. To ensure consistency across different

documents and data sources, naming normalization is performed before the information is consolidated into a structured safety analysis schema.

The resulting schema serves as an intermediate engineering model that explicitly represents functions, configurations, systems, components, and associated failure modes. Because the schema is generated independently of the final fault tree structure, it can be reviewed and refined by domain experts prior to fault tree construction. This review process enables the correction of missing information, modification of relationships, and validation of extracted knowledge before it is used for safety analysis.

After the review process is completed, the finalized schema is transformed into a fault tree model. During this transformation, the relationships captured in the schema are used to define fault propagation paths, intermediate events, basic events, and logical gate structures. By separating knowledge construction from fault tree generation, the proposed framework improves traceability, consistency, and transparency throughout the modeling process while providing opportunities for intermediate validation and expert intervention.

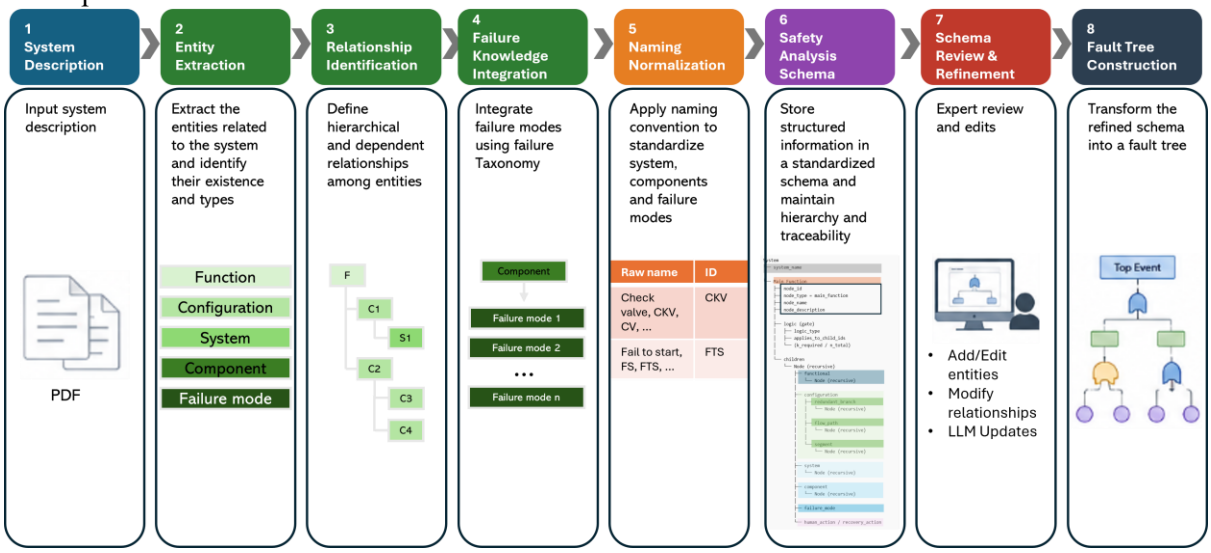

Figure 1: Overall workflow of the proposed schema-guided fault tree generation framework

2.2. Knowledge Model Construction

The knowledge model required for fault tree generation is constructed incrementally from a system description.

2.2.1. Entity Extraction

The first step in knowledge model construction is the identification of engineering entities contained within the system description. These entities represent different levels of abstraction within the system and provide the basis for subsequent knowledge modeling. For example, consider the following system description “The auxiliary cooling system removes decay heat through two redundant cooling trains. Each train contains a pump and a heat exchanger”. From this description, the model identifies the safety function of removing decay heat, recognizes the existence of two redundant cooling train configurations, determines that the auxiliary cooling system performs the function, and extracts the associated physical components such as pumps and heat exchangers. At this stage, the objective is limited to identifying and classifying relevant entities. No assumptions are made regarding how these entities are connected or how failures propagate through the system. By separating entity identification from relationship reasoning, the framework reduces the complexity of the extraction task and improves the consistency of the resulting knowledge model.

2.2.2. Relationship Identification

Once the relevant entities have been identified, the next step is to establish relationships among them. Fault tree construction requires more than a collection of isolated entities; it requires an explicit representation of how functions depend on configurations, how configurations are realized through systems, and how systems are composed of individual components.

Using the entities extracted in the previous stage, the framework constructs a hierarchical representation that captures both structural and functional dependencies. For example, the function of decay heat removal may be linked to a specific cooling train configuration, which in turn is associated with the auxiliary cooling system and its constituent components. Through this process, the framework transforms a set of independent entities into a coherent engineering model that reflects the operational structure of the system.
The resulting relationships provide the foundation for subsequent fault tree generation because they define the pathways through which failures can propagate from component-level events to higher-level system and functional failures.

2.2.3. Failure Knowledge Integration

After the system structure has been established, failure-related information is incorporated into the knowledge model. Instead of allowing the language model to freely generate failure modes, this study utilizes a predefined failure taxonomy derived from existing fault tree examples. This approach ensures consistency in failure mode representation and reduces variability caused by differences in terminology across engineering documents.
For each identified component, applicable failure modes are selected from the taxonomy and linked to the corresponding component. For example, a pump may be associated with failure modes such as fail to start, and fail to run, while a valve may be associated with fail to open, and fail to close. By systematically connecting components with standardized failure modes, the framework establishes the basic events required for fault tree construction.

2.2.4. Naming Convention and Normalization

To support terminology normalization, fault tree examples are analyzed to identify recurring naming patterns and synonymous expressions, which are then organized into a naming convention database. This database contains mappings between alternative component names, equipment tags, functional descriptions, and standardized failure mode terminology. Using the provided naming rules and terminology mappings, the LLM aligns semantically equivalent entities with standardized representations and retrieves relevant fault tree patterns from the database. This allows pattern retrieval to be performed based on semantic similarity and engineering meaning rather than exact keyword matching. The retrieved patterns provide reusable engineering knowledge that can be linked to the extracted knowledge model and subsequently utilized during fault tree construction.

2.3 Safety Analysis Schema

The resulting knowledge model is stored in the form of a safety analysis schema. The schema provides a structured representation of engineering knowledge by explicitly modeling functions, configurations, systems, components, failure modes, and, when necessary, human actions.
The function represents the safety objective or operational purpose of the system, such as "Decay Heat Removal" or "Emergency Core Cooling." The configuration represents the arrangement or operational configuration required to achieve a particular function. Examples include redundant trains and flow paths. The system represents a collection of component required to perform a specific function, while the component represents a physical element such as a pump, valve, heat exchanger, or power supply unit. The failure mode represents a potential failure event associated with a component, such as fail to start, fail to run, fail to open, or internal leakage.
In certain cases, higher-level functional failures or human action failures may also contribute directly to the top event. Such events can also be represented within the schema. Although the schema generally follows a hierarchical organization, it also supports cross-level functional and dependency relationships when required by system characteristics. A typical schema structure is shown in Figure 2.

Figure 2: The Safety Analysis Schema

Each entity in the schema contains attributes such as a unique identifier, node type, description, and logical relationships. Parent-child relationships allow recursive expansion of the schema, enabling representation of complex system architectures and multi-level functional dependencies.

By providing a flexible and extensible representation of engineering knowledge, the schema serves as a standardized intermediate representation for fault tree generation while maintaining traceability and consistency throughout the modeling process.

2.4 Schema Review and Refinement

The generated schema serves as an intermediate representation that can be reviewed and refined before fault tree construction. Unlike conventional end-to-end generation approaches, the proposed framework allows users to inspect the extracted engineering knowledge and verify its consistency prior to transforming it into a fault tree model.

To support this process, the schema is stored in a structured format and visualized through a web-based interface as illustrated in Figure 3. Users can examine the hierarchical relationships among the entities, enabling them to trace how a particular failure mode is connected to higher-level systems and safety functions. This visualization improves transparency and helps identify missing entities, incorrect relationships, or inconsistencies in the generated knowledge model.

Within the Explorer/Hierarchy interface, users can directly review and modify individual nodes and their relationships. For each node, analysts may decide whether the information should be included, excluded, or flagged for further review. Additional systems, components, configurations, and failure modes can be added manually, while incorrect relationships can be corrected or removed.

The chat interface serves as an evidence-tracing assistant. It enables analysts to query where a particular entity appears across intermediate workflow artifacts and to verify whether that information is propagated into the integrated fault tree output. Actual modifications to the reviewed model are performed through the structured the explorer interface.

After the review process is completed, the selected nodes and relationships are saved as a user-approved integrated JSON file. This reviewed JSON serves as the authoritative input for the subsequent fault tree generation stage. The saved JSON is then converted into fault tree artifacts, including fault tree logic (FTL), gate description file (GTD), and basic event definitions (BED), which are used to construct the final fault tree model. This review process provides an explicit human-validation step that helps mitigate extraction errors and unsupported model assumptions before fault tree generation.

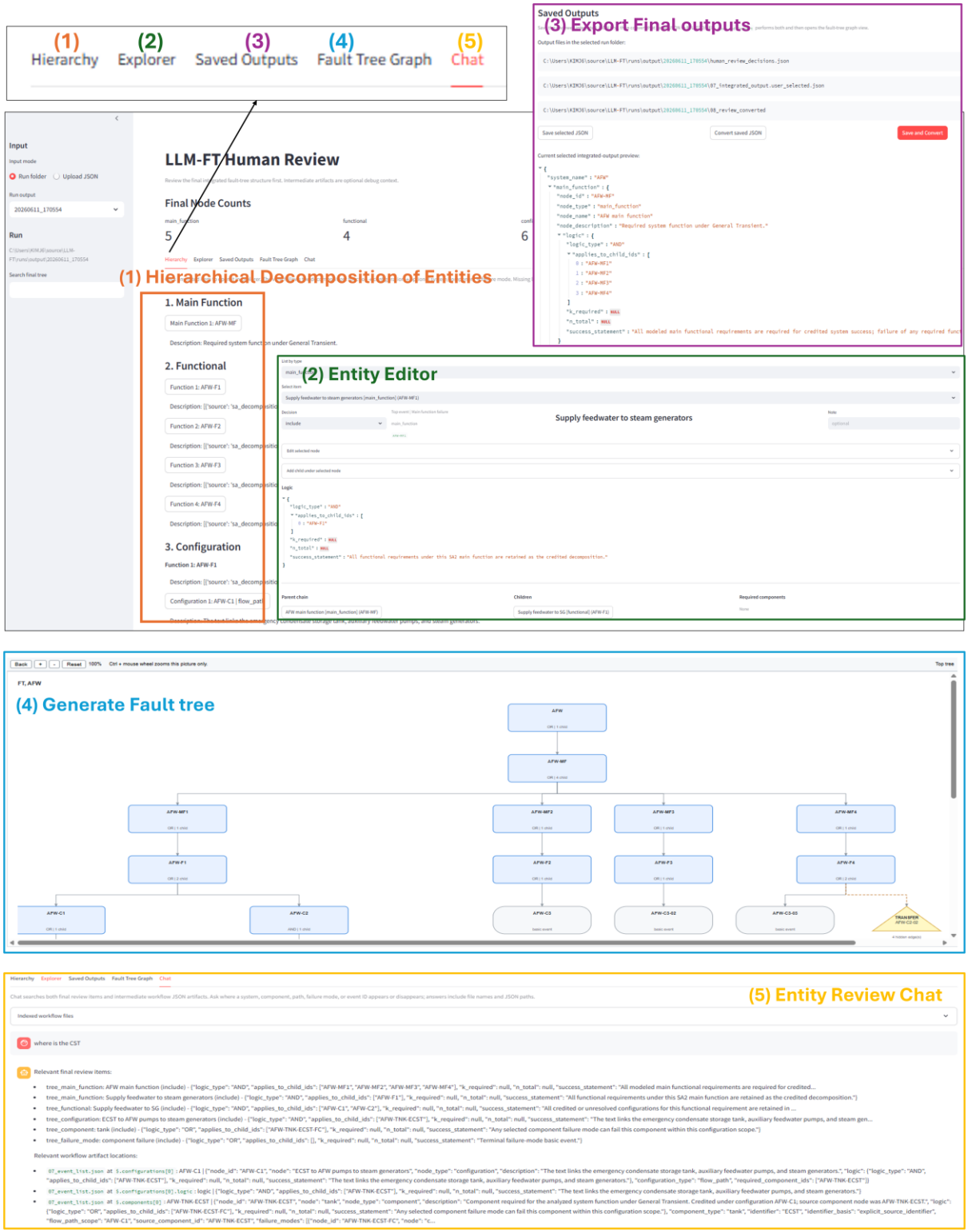

Figure 3: User Interface for Schema Review and Refinement

2.5 Transformation to a Fault Tree Model

Once the schema review process is completed, the finalized safety analysis schema serves as the input for fault tree construction. The transformation process utilizes functions, configurations, systems, components, and failure modes to generate top events, intermediate events, and basic events. Logical gates such as AND and OR are then established according to system configurations and dependency relationships. During this process, the success-oriented system representation captured in the schema is converted into a failure-oriented representation suitable for fault tree modeling. This transformation is

essential because system descriptions and configurations are typically defined in terms of successful operation, whereas fault trees represent the conditions leading to functional failure.
For example, a flow path may consist of several components connected in series. From a success perspective, the configuration is achieved only when all components in the flow path are available and functioning. However, from a failure perspective, the flow path fails if any one of the components fails. Consequently, the corresponding fault tree structure is represented using an OR gate connecting the component failure events.
Similarly, redundant configurations that require only one train or subsystem to succeed are transformed into failure logic in which all redundant paths must fail before the associated function is lost. In such cases, AND gates are used to represent the loss-of-function condition. Through this conversion from success logic to failure logic, the fault tree accurately captures the causal relationships leading to system and functional failures.

3. MINIMAL CUT SET-BASED FAULT TREE EVALUATION

The logical correctness of a generated fault tree can be evaluated by comparison with a reference fault tree. However, fault trees developed by different analysts may vary in their structure, level of decomposition, or event organization even when they represent the same failure logic. As a result, generated and reference fault trees may not match exactly even when they describe similar failure mechanisms. Therefore, direct structural comparison may not be an appropriate evaluation method because different fault tree structures may represent the same logical relationship.
This study first evaluates how closely the generated fault tree reproduces the logical conditions represented in the reference model by comparing their minimal cut sets (MCSs). Since an MCS represents the minimum combination of basic events required to cause the top event, MCS comparison provides a practical measure of logical equivalence between the two models. The comparison is quantified using precision, recall, and F1 score based on exact MCS matching, providing objective measures of logical equivalence between the generated and reference models. In addition, because partial differences may still exist even when exact matches are not obtained, similar MCSs are identified using Jaccard similarity and further analyzed to identify potential errors and understand the sources of disagreement between the generated and reference fault trees.

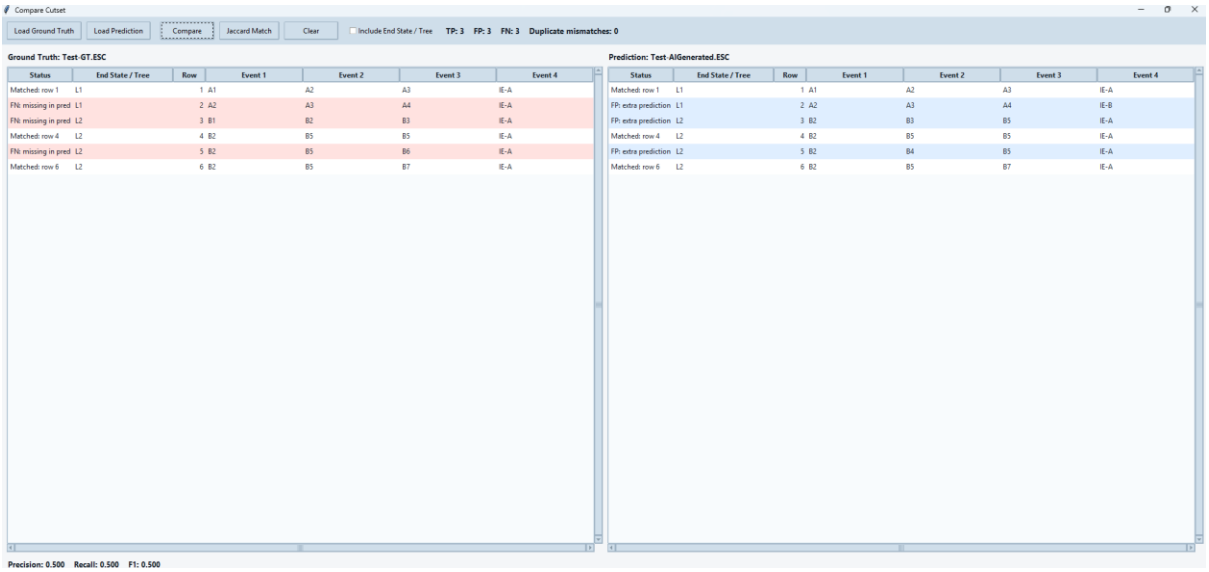

Figure 4: MCS Comparison between Reference and Generated Fault Tree Models

To evaluate the logical equivalence of fault trees, MCSs first have to be extracted from both the reference fault tree and the generated fault tree. However, identical events may be represented using different names across models, making direct comparison difficult. Therefore, a normalization process is required prior to MCS comparison. In this study, a predefined naming convention is applied to standardize event names in both the reference and generated models before MCS generation. The

normalized MCSs are represented as sets of basic events, and the ordering of events within a cut set does not affect the comparison. Consequently, cut sets composed of the same combination of events are regarded as identical regardless of the order in which the events are listed. This approach eliminates differences arising from alternative fault tree representations and enables comparison based solely on the fundamental logical conditions that lead to the top event.
Once the MCSs have been generated and normalized, the reference and generated MCS sets are compared to identify matching and non-matching cut sets. Figure 4 illustrates the MCS comparison process used to identify matching, missing, and extra cut sets between the reference and generated fault tree models. Through this process, it is possible to evaluate how accurately the generated fault tree reproduces the failure logic represented in the reference model.

The set of MCSs extracted from the reference fault tree is treated as the ground truth, while the set extracted from the generated fault tree is treated as the prediction. A cut set appearing in both sets is classified as a true positive (TP). A cut set appearing only in the generated model is classified as a false positive (FP), whereas a cut set appearing only in the reference model is classified as a false negative (FN). Based on these definitions, the following evaluation metrics are calculated:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3}$$

Precision measures the proportion of generated cut sets that are logically correct, indicating the extent to which unnecessary failure conditions are introduced. Recall measures the proportion of reference cut sets successfully reproduced by the generated model, indicating how completely the required failure logic is captured. The F1 score provides a balanced measure of overall logical agreement between the generated and reference Fault Trees.

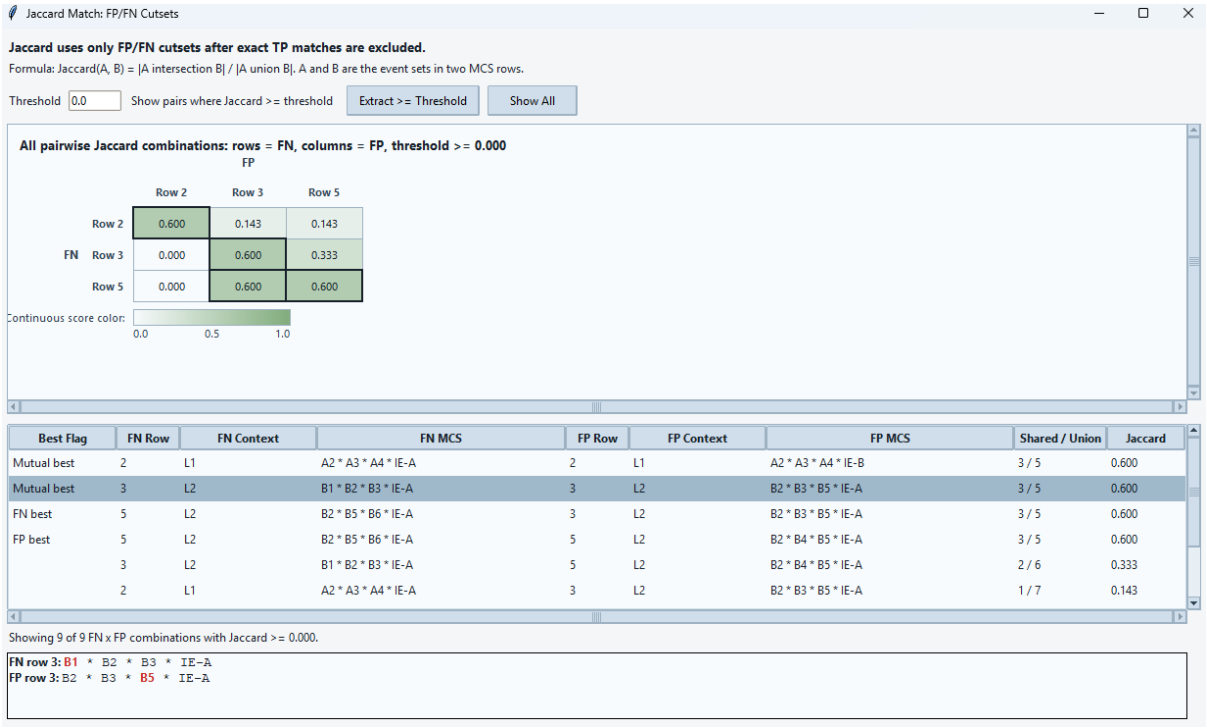

Figure 5: Partial Matching Analysis of Unmatched Minimal Cut Sets Using Jaccard Similarity

Precision, recall, and F1 score are based on exact MCS matching and therefore provide a quantitative assessment of logical correctness. However, exact matching alone does not reveal how close an unmatched generated cut set is to a missing reference cut set. To support more detailed analysis, this study performs Jaccard similarity-based partial matching for unmatched MCSs. Jaccard similarity is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4}$$

where (A) and (B) represent two cut sets.

The purpose of this analysis is not to modify the TP, FP, FN, Precision, Recall, or F1 calculations. Instead, Jaccard similarity is used to identify the most similar cut set pairs among unmatched reference and generated MCSs. Figure 5 illustrates the Jaccard similarity-based comparison process. Higher similarity scores indicate greater overlap between two cut sets and suggest that the generated model has captured a substantial portion of the intended failure logic despite failing to achieve an exact match.
For each unmatched cut set, the most similar reference-generated pair can be examined to identify events that are present in one cut set but absent in the other. These differences provide diagnostic information that can help analysts investigate potential omissions, extra failure conditions, or modelling discrepancies in the generated fault tree. By supporting targeted review of logical differences, the Jaccard similarity analysis provides additional insight into the nature of disagreements between generated and reference fault trees.

4. CONCLUSION AND FUTURE WORK

This paper presented a schema-guided structured reasoning framework for fault tree generation from system descriptions. Rather than generating fault trees directly from textual inputs, the proposed approach constructs a safety analysis schema as an intermediate representation that organizes engineering knowledge into functions, configurations, systems, components, and failure modes. By separating knowledge construction from fault tree generation, the framework improves traceability, consistency, and transparency throughout the modeling process.
The proposed framework also incorporates a human-reviewable workflow that enables analysts to inspect, validate, and refine generated engineering knowledge prior to fault tree construction. The reviewed schema is subsequently transformed into fault tree artifacts through structured logic conversion rules. In addition, an MCS-based evaluation methodology was introduced to assess logical agreement between generated and reference fault trees. By combining exact MCS matching with Jaccard similarity-based analysis, the proposed approach supports both quantitative evaluation and diagnostic review of logical discrepancies.
Future work will focus on applying the framework to a set of nuclear safety analysis case studies and evaluating the generated fault trees against expert-developed reference models. Additional research will investigate automated consistency checking across intermediate artifacts and enhanced reasoning strategies for complex system configurations. The framework will also be extended through integration with additional engineering information sources, including P&ID analysis agents and knowledge graph-based agents, enabling fault tree generation to leverage both document-derived and structured engineering knowledge. Ultimately, these capabilities will be incorporated into a broader multi-agent safety analysis architecture supporting risk assessment, safety model development, and engineering decision-making workflows.

References

[1] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, “Fault Tree Handbook,” U.S. Nuclear Regulatory Commission, NUREG-0492, Washington, DC, USA, (1981).

[2] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick III, and J. Railsback, “Fault Tree Handbook with Aerospace Applications, Version 1.1,” NASA Office of Safety and Mission Assurance, Washington, DC, USA, (2002).
[3] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz and D. Roth, “Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey”, ACM Computing Surveys, vol. 56, no. 2, 30, (2023).
[4] H. Sun, M. Hauptman and R. Lutz, “Integrating Product-Line Fault Tree Analysis into AADL Models,” Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium (HASE), pp. 15-22, (2007).
[5] F. Mhenni, N. Nguyen and J.-Y. Choley, “Automatic Fault Tree Generation From SysML System Models,” Proceedings of the IEEE International Symposium on Systems Engineering (ISSE), (2014).
[6] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith and D. C. Schmidt, “A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT”, arXiv preprint arXiv:2302.11382, (2023).
[7] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto and P. Fung, “Survey of Hallucination in Natural Language Generation”, ACM Computing Surveys, vol. 55, no. 12, 248, (2023).