An Evaluation Framework for Fault Diagnosis Using Technical Manuals in Retrieval-Augmented LLMs

Sarah Lukens[1], Matthew Bishof[1], Nadir Siddiqui[1], Destiny West[1],

[1] *LMI, Tysons, VA, 22102, USA*
sarah.lukens@LMIsolutions.com, mbishof@LMIsolutions.com, nadir.siddiqui@LMIsolutions.com, destiny.west@LMIsolutions.com

ABSTRACT

Fault diagnosis is a time-intensive maintenance task often reliant on the expertise of senior technicians. As this workforce ages and demand grows for digital tools, there is a growing need to capture and automate this knowledge while maintaining the precision required for technical applications. This study introduces an evaluation-driven framework for fault code recommendation, applied to a ground vehicle diagnosis system. Two tasks were designed to reflect potential system configurations: (1) a chat-style task simulating large language model (LLM) interaction, and (2) a label-constrained task using structured fault codes from technical manuals. Multiple retrieval-augmented generation (RAG) configurations were compared against LLM-only and retrieval-only baselines. Results showed that retrieval-based methods outperformed LLM-based ones for label-matching tasks, while the chat task showed challenges in linking observations to fault codes from the manual. These results highlight the importance of aligning task design with evaluation goals and considering retrieval-first approaches as viable alternatives to LLMs in technical language processing (TLP) applications. Beyond experimental findings, we outline industrial lessons learned: the importance of aligning system design to use case goals, adopting evaluation-first validation, and the need to pilot LLM-based systems under realistic conditions. These lessons provide practical guidance for developing effective diagnostic support systems in industrial contexts.

1. INTRODUCTION

Troubleshooting is often the most time-consuming and skill-dependent phase of the maintenance process, requiring personnel to diagnose interrelated faults across complex systems (Shin, Tien, & Prabhu, 2019), (Schaafstal, Schraagen, & Van Berl, 2000). Much of this process depends on use of lengthy technical manuals, where operators and maintainers must manually cross-reference observed symptoms with fault codes and procedures. The challenge is compounded by an aging workforce, where much of the expertise is tacit and acquired through years of hands-on practice. As experienced personnel retire, organizations face the dual pressures of preserving institutional knowledge and equipping a new generation of maintainers who expect modern, digitally assisted tools.

Advances in artificial intelligence (AI), and particularly large language models (LLMs), create new opportunities to design more intuitive and responsive diagnostic support systems. Retrieval-based methods for fault diagnosis are effective at surfacing relevant technical content and provide an auditable, established baseline. However, they can be limited for complex cases or enabling interactive dialogue between humans and structured procedures. LLMs, by contrast, offer the flexibility to synthesize information across disparate sources, interpret unstructured natural-language input, and generate interactive outputs that adapt to user prompts. Early demonstrations show that LLMs can produce plausible fault codes, but an open challenge is in designing and validating such systems in ways that build trust and deliver measurable value in field settings.

In a previous study, we developed a workflow and self-evaluation framework for prognostics and health management (PHM) that automated maintenance recommendations triggered by PHM alerts (Lukens, McCabe, Gen, & Ali, 2024). A significant finding from that study was that validation by subject matter expert was prohibitively time and resource intensive. Challenges in efficiently measuring performance impeded our ability to conduct deeper explorations, such as integrating technical manuals into a retrieval-augmented generation (RAG) system design to inform a troubleshooting recommender agent. From this, we identified performance evaluation in the design loop as a critical design requirement for the design-phase of such as system.

In this study, we extend the previous work by developing an evaluation-first approach towards integrating technical manuals into fault diagnosis workflows. More broadly, we consider the process of how to design and validate diagnostic

support systems in ways that are trustworthy and effective in field settings. Using this approach, we show how structured evaluation can inform system design and reveal tradeoffs between retrieval-based and generative methods. To illustrate this, we use a narrowly scoped dataset synthesized from technical manuals. While the data itself is synthetic, the approach is transferable to proprietary or classified datasets, making the findings relevant for real-world solution development. Beyond the dataset, we also consider the practical challenges of handling technical manuals, which often contain complex elements such as diagrams, nested tables, and extensive cross-referencing which can complicate the extraction and organization of information that LLMs can interpret reliably.

Our main contributions are as follows:

- We introduce an evaluation-driven fault code recommendation system that integrates structured technical manual content into both retrieval-based and large language model (LLM) configurations.
- Through two constrasting tasks (chat-style versus label-constrained), we demonstrate how task framing and evaluation choices affect observed performance, and when retrieval may outperform LLMs.
- Industrial lessons learned. Experimental findings are translated into practical guidance for industrial system design for using LLMs.

The remainder of this work is organized as follows. Section 2 covers background and related work, covering an overview of the evolution of intelligent fault diagnosis systems, emerging LLM concepts, related work using LLMs in fault diagnosis and troubleshooting systems and performance evaluation. In Section 3, we describe the methodology and case study for the fault recommender system. In Section 4, we introduce the first task with quantitative and quantitative performance insights. The lessons learned segue to Section 5, where we introduce the second task with its performance insights. A discussion of the results along with limitations and industrial lessons learned are covered in Section 6.

2. BACKGROUND AND RELATED WORK

Troubleshooting is a central task in maintenance, where an operator or technician begins with an observed symptom and must narrow down the possible faults that could explain it. For example, an operator may observe and report "engine shaking a lot while driving", prompting the need to identify candidate faults and locate corresponding troubleshooting procedures in the manual.

In the broader health monitoring context, a complete PHM system includes the following components: (1) data collection; (2) condition assessment or health modeling capabilities; (3) decision-making logic for initiating actions based on system state; and (4) feedback mechanisms to validate or refine predictions. Fault recommendation links condition assessment to decision support by mapping condition indicators or operator observations to plausible faults, guiding the application of troubleshooting procedures and corrective actions. These activities are often time-sensitive and rely heavily on technician expertise and access to accurate documentation.

Fault diagnosis involves detecting, isolating and identifying impending or incipient failure conditions (Vachtsevanos, Lewis, Roemer, Hess, & Wu, 2006). Intelligent fault diagnosis refers to using computational tools, such as machine learning algorithms, to aid in these tasks. Fault recommendation, as a subset, generates candidate faults and their associated procedures in response to observations, which can be a component of an intelligent fault diagnosis system that offers interactive, recommendation, and multi-turn chat capabilities.

2.1. Intelligent Fault Diagnosis Systems

Early approaches to automated fault diagnosis were largely based on Rule-based and Case-based Reasoning (CBR). Rule-based systems rely on logic trees or predefined sets of instructions to guide users through fault isolation procedures. CBR extends these ideas by leveraging historical fault and repair data, generating recommendations based on prior cases. Both Rule-based and CBR methods are transparent to audit, which makes their evaluation relatively straightforward. However, Rule-based systems often lack flexibility in handling complex faults or edge cases, while CBR systems can struggle to generalize beyond the scenarios represented in their databases.

Data-driven and AI-based methods for fault diagnosis can be broadly categorized into Retrieval-based and Generative approaches. Retrieval-based systems return relevant documents or responses from a fixed knowledge base, often using keyword matching or similarity measures such as TF-IDF or BM25. These methods are efficient, predictable, and easy to evaluate using standard metrics such as precision and recall on ranked outputs. Their main limitation, however, is that they are constrained to the information already encoded in the system (Xie, Tang, Gu, & Cui, 2025).

Generative methods, particularly those enabled by LLMs, introduce a dialogue layer capable of synthesizing information across multiple sources. Generative models show promise for handling edge cases and enabling interactive troubleshooting, but also raise challenges around hallucination, inconsistency, and the difficulty of measuring and controlling performance (Azevedo et al., 2023). Evaluating generative models is challenging because correctness is not always the lexical overall but rather the relevance of the generated content to the scenario.

While retrieval-based systems are auditable and predictable,

LLM-based generative systems offer flexibility and interactional capabilities in addressing complex fault scenarios. Recent studies and controlled experiments have highlighted the costs and complexities involved in assessing the performance of these systems, especially in multi-turn dialogue scenarios (Löwhagen, Schwendener, & Netland, 2025)(Alghamdi, Halvey, & Nicol, 2024).

2.2. Integration of Structured Knowledge

Structured knowledge representation is central to making technical content usable by diagnostic systems. KGs and ontologies provide an organized way to encode information such as maintenance data, fault codes, and troubleshooting procedures into machine-readable formats that support reasoning and retrieval. There has been considerable work in industry focused on developing ontologies for the maintenance domain, design for interoperability and standardized, reusable resources that others can adopt (Rajpathak, 2013; Karray, Ameri, Hodkiewicz, & Louge, 2019; Hodkiewicz, Klüwer, Woods, Smoker, & Low, 2021).

Several recent studies have explored using LLMs as tools to assist in building and populating domain-specific ontologies. Examples include ontologies for maintenance actions derived from historical records using Technical Language Processing (TLP) and LLM-based extraction of procedures from manuals (Woods, Selway, Bikaun, Stumptner, & Hodkiewicz, 2023; Woods, French, Hodkiewicz, & Bikaun, 2023). LLMs have also been applied to transform unstructured manuals into structured troubleshooting workflows, linking assets to symptoms, causes, and repair steps (Vidyaratne et al., 2024). KG construction has likewise been demonstrated in industrial contexts, including aircraft fault diagnosis (Tang et al., 2023) and equipment fault graph methodologies (Xie et al., 2025). Specific applications in aviation have shown that KG-based approaches can improve diagnostic accuracy and interpretability (Peng & Yang, 2024; Meng et al., 2023).

These structured frameworks provide a foundation for integrating LLM-augmented generative reasoning into fault diagnosis systems, typically drawing on technical manuals, historical data, and other contextual sources. Technical manuals, however, are often distributed as PDFs that capture information in unstructured or semi-structured forms. Their inconsistent formatting, such as mixing text, images, and structured layouts, often leads to parsing erros, information loss, and degraded input quality that can propagate into downstream applications. Recent evidence shows that extraction strategies materially affect retrieval and LLM performance: for example, (Adhikari & Agarwal, 2024) compared PDF parsing tools across document types and tasks, demonstrating the importance of selecting appropriate parsing methods. Ultimately, the reliability of knowledge graphs in fault diagnosis systems depends not only on their construction, but also on how effectively they can be integrated into end-user workflows.

2.3. Emerging LLM-augmented approaches

Recent advancements in adapting LLMs for technical applications incorporate approaches such as RAG, fine-tuning, domain-specific data augmentation, multimodal integration, and hybrid designs. RAG enhances LLMs with external knowledge sources to improve the accuracy and relevance of generated text (Gao et al., 2023; Lewis et al., 2020). Classic RAG approaches typically store unstructured text in vector databases, but challenges such as the "context killer" problem, where important information is fragmented across chunks, can lead to incomplete or incorrect answers. Approaches like GraphRAG address this by incorporating graph-structured knowledge to support reasoning over complex technical relationships (Edge et al., 2024).

Fine-tuning and data augmentation offer complementary strategies for domain adaptation. Fine-tuning improves performance by training LLMs on specialized vocabularies for tasks such as fault diagnosis and maintenance troubleshooting (Lu, Luu, & Buehler, 2025), but fine-tuned models do not generalize well across domains or applications and are computationally intensive. Domain-specific data augmentation expands training sets through synthetic generation, adaptation of related-domain data, or transformation of existing data to better reflect operational scenarios (Jadon, Patil, & Kumar, 2025). This introduces additional complexity in orchestrating evaluation across modules.

Beyond text-based strategies, agent-based systems augment LLMs with specialized agents that provide decision support in response to specific triggers, while multimodal models extend capabilities by integrating text with other modalities such as diagrams, tables, or images, enabling use of the full range of technical documentation formats (Liang et al., 2024). Hybrid approaches aim to combine the strengths of these methods. For example, HybridRAG stores unstructured data in a vector database while representing digital manual content in knowledge graphs, creating retrieval-based fault diagnosis systems (Xie et al., 2025). Challenges remain in how to evaluate performance when dialogue layers introduce interactional variables, making user trials essential beyond the design phase. For instance, (Löwhagen et al., 2025) demonstrated benefits of AI assistants in supporting less experienced technicians.

These approaches are increasingly realized in application contexts. Several studies focus on root cause recommendation tasks that rely on text data. Troubleshooting systems employ LLMs to infer root causes from unstructured reports (Trilla, Yiboe, Mijatovic, & Vitri à, 2024), FaultExplainer generates diagnostics and root cause analysis for chemical processing operators (Khan, Nahar, Chen, Flores,

& Li, 2025), and pre-trained LLMs have been benchmarked across data and feature variations for fault diagnosis (Zheng, Pan, Liu, & Chen, 2024). Customized fine-tuned models have been developed for maintenance Q&A applications (Chen, Tian, Zhang, Li, & Zhang, 2025), while systems like DefectTwin integrate LLMs with digital twins for visual inspection of railway defects (Ferdousi, Hossain, Yang, & Saddik, 2024).

Extending beyond text, LLMs are being adapted to numerical or multimodal inputs. Methods for labeling data using Technical Language Supervision fuse textual and sensor data (Löwenmark, 2025; Löwenmark, Taal, Schnabel, Liwicki, & Sandin, 2021), while reasoning agents have been applied to reduce false alarms in condition monitoring systems (Löwenmark, Strömbergsson, Liu, Liwicki, & Sandin, 2025). Deep Root Cause Analysis integrates time-series data with multi-LLM debate strategies (Huang, Shah, Karigiannis, & Evans, 2024), and MaintAGT combines signal-to-text models with physical knowledge for fault diagnosis (He et al., 2024). In safety-critical contexts, LLMs integrated with anomaly detection models enable explainable fault diagnosis in nuclear power plants (Dave, Nguyen, & Vilim, 2024), while other work applies multimodal methods to vibration-based diagnosis (Qaid, Zhang, Li, Ng, & Li, 2024), bearing fault detection (Tao et al., 2025), and turbomachinery root cause analysis (Vitale et al., 2024).

2.4. Performance Evaluation

Performance evaluation is essential for developing reliable AI-augmented diagnostic systems. In industrial settings, trust must be established before deployment. Waiting until roll out to assess accuracy or usability risks system failures and can lose operator trust which can impede a tool's adoption. Conducting evaluation during the design phase enables iterative improvement, shape system choices and ensuring robustness. Rather than relying on visually appealing demonstrations of toy prototypes, an evaluation-first approach integrates rigorous performance measurement into the development process, using structured tests and feedback loops to advance from early prototypes to reliable production systems.

Prior work has emphasized the importance of user-facing assessments, such as controlled experiments with AI assistants in dialogue settings that capture cognitive and interactional variables (Löwhagen et al., 2025). Our focus is earlier in the lifecycle: building structured, repeatable methods to determine whether a design is on track before large-scale user trials. For LLM- and RAG-based diagnostic systems, design-phase evaluation approaches can be grouped into three main categories: retrieval evaluation, generation evaluation, and LLM-as-judge/hybrid evaluation.

For retrieval evaluation, metrics such as Hit Rate, Precision, Recall, and Mean Reciprocal Rank quantify how effectively relevant documents are surfaced. Retrieval evaluation often relies on established algorithms like TF-IDF and BM25, which use term frequency and inverse document frequency or probabilistic weighting to rank documents based on query relevance. These classic methods are interpretable and provide a robust baseline for comparison against more advanced LLM-augmented retrieval systems. However, while these metrics and algorithms are well-understood, they may not capture "partial credit" scenarios where multiple related fault codes are valid, limiting their ability to fully reflect a system's practical usefulness.

For generation evaluation, metrics like BLEU, ROUGE, and F1-based scores compare generated outputs against reference labels (Doris et al., 2024). They offer repeatable overlap measures but often fail to capture adequacy, completeness, or contextual accuracy (Zhu et al., 2024). For example, a generated response may achieve high n-gram overlap yet still lack the logical coherence expected by human evaluators.

For LLM-as-judge and hybrid evaluation, using LLMs to grade outputs provides scalable ways to capture semantic correctness beyond lexical overlap and can support tasks such as error checking or annotation. However, risks include bias and omission of critical content if systems are left unchecked. In practice, hybrid approaches that combine automated scoring with human review balance the scalability of machine evaluation with the domain expertise required for reliable assessment.

3. METHODOLOGY

3.1. Fault Recommender System

The LLM-Augmented Fault Recommender System used in this study is illustrated in Figure 1. It consists of two main modules: the Maintenance Troubleshooting Module and the Knowledge Processing Module. The Maintenance Troubleshooting module is returns a recommended list of possible faults in response to the input of an operator observation. The Knowledge Processing Module contains structured representations of technical information extracted from manuals and other knowledge sources, such as maintenance history data.

3.2. Case Study

The case study focuses on exploring LLM systems to augment how operators interact with equipment manuals to identify and resolve observed issues. Operators may encounter faults either during routine preventative checks or while operating the equipment. In practice, mapping these observed behaviors to fault codes in a manual can be challenging. A single symptom may correspond to multiple potential faults, and the manuals themselves are often lengthy and complex, sometimes spanning hundreds of pages. As a result, the current process typically requires operators to search through

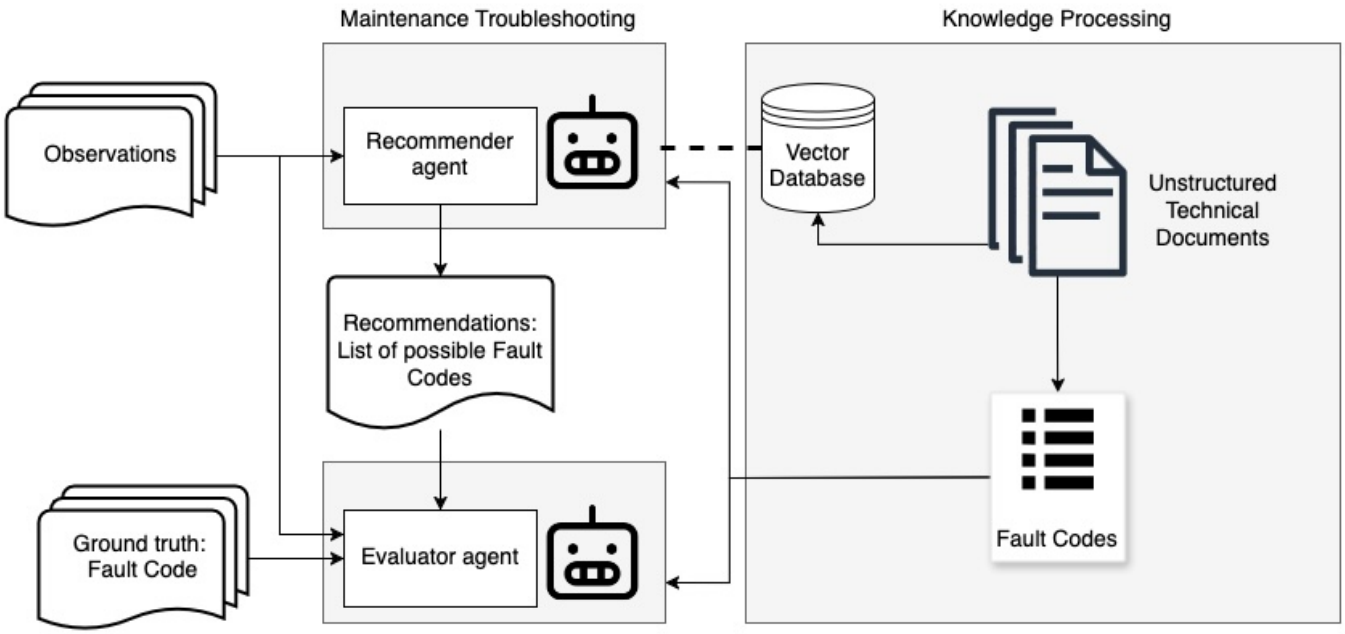

Figure 1. Cartoon of the full Fault Recommender System in this study which consists of two modules. The Maintenance Troubleshooting module returns recommended list of possible faults in response to an input operator observation. The Knowledge Processing module contains digitized content such as from technical manuals the Maintenance Troubleshooting module can use for context.

large amounts of documentation to locate the appropriate troubleshooting procedure for a given symptom.

The case study is based on a non-classified manual from 1998 for light medium tactical vehicles (LMTV), which is available online (U.S. Department of the Army, 1998). Although this manual is outdated and unauthorized, its standardized structure allows us to adapt the approaches developed around it to other, more current and sensitive manuals, particularly those concerning ground systems. MIL-STD-3031, a military standard on Army business rules for technical manuals, outlines various components, including fault codes (malfunctions) and troubleshooting procedures for operators (U.S. Department of the Army, 2008). This adherence to standardized formats like MIL-STD-3031 demonstrates the potential for generalizing techniques across different manuals within the Army.

We specifically use the "9-2320-365-10" manual, which is an operator's manual for LMTVs. The manual contains a malfunction list of 224 malfunctions within 22 systems along with the operator troubleshooting procedures for each malfunction.

Observation data. The data used in the case study is synthetically generated using an LLM to emulate a collection of historical observations by operators with the actual fault code. The data is generated by randomly sampling fault codes with replacement, providing the LLM with text from the manual as context, and instructing the LLM to generate synthetic unstructured potential operator observations with rationale. A sample of a few generated observations and their rationales along with the fault code are shown in Table 1.

Although in this setup an observation is generated from a selected fault code, it is noted that in industrial systems an observed fault can have multiple potential causes or malfunctions. This means that multiple fault codes may be possible from one observation. For this reason, the recommendation system produces multiple possible codes, but we assume that for performance the true fault should be included among its suggestions.

There are documented risks associated with using LLMs to generate synthetic data. These include the "ceiling effect", where future systems may be constrained by the limitations of the generating model and the potential risk of undermining future system evaluation when such systems surpass the generating model capabilities (Soboroff, 2025). To mitigate these risks in the case study, a more advanced LLM was used to generate the data than the one used in the troubleshooting system. 300 observations were synthetically generated using OpenAI's *gpt-4.1* endpoint, while the implementation of the Recommender and Evaluation agents relies on OpenAI's *gpt-3.5-turbo-0125* endpoint (Achiam et al., 2023). We note that the modular framework allows for alternative models.

Table 1. Sample of 3 synthetically generated historical observations with "true" observed fault code. The middle column provides a rationale by the LLM for the observation to help validate the observation.

Observed by Operator	Rationale	(True) Fault Code
truck feels sluggish and struggles to accelerate even when pressing the gas pedal	A noticeable lack of acceleration and sluggish performance is consistent with 'LOW ENGINE POWER,' which could be caused by fuel contamination, restricted air filters, low engine oil, or air intake issues as outlined in the troubleshooting procedures.	LOW ENGINE POWER.
fuel gauge does not light up when dashboard lights are turned on	If the fuel gauge fails to illuminate when the operator turns on the panel lights, it may indicate a burned-out bulb, faulty wiring, or switch position issue as covered in the troubleshooting procedure.	FUEL GAUGE DOES NOT ILLUMINATE.
a lot of water comes out when draining air tanks	If the operator notices an unusually large amount of water being expelled during the draining of the air reservoirs, it indicates excessive moisture buildup within the compressed air system.	LARGE QUANTITY OF MOISTURE EXPELLED FROM AIR RESERVOIRS.

3.3. Implementation

In implementation, the agents are LLMs which are induced to return structured recommendation or evaluation objects using function-calling (Eleti & Kilpatrick, 2023). For this study, the system scope is limited to the Recommender Agent, which is responsible for generating a list of fault codes that plausibly explain a given operator observation. For each observation, the Recommender Agent is given the following prompt:

> You are an operator of LMTV trucks in the Army, and you have observed the following issue(s): [observed], pertaining to the asset [asset]. Write a list of [step number] possible fault codes or malfunctions which may contribute to the observed behavior. Your answer should contain only the fault codes and nothing else.

The Evaluator Agent then compares each recommended fault code against historical ground-truth data, assigning a Boolean label of True if the recommendation is correct, or False otherwise. The Evaluator Agent is given the following prompt:

> You are an operator of LMTV trucks in the Army, and you have observed the following issues(s): [observed], pertaining to the asset [asset]. This prompted a thorough manual investigation, revealing the following fault: [failure_mode]. Without knowing the true fault, the following list of possible malfunctions were proposed: [plan]. For each suggested fault in the list, assess whether or not this fault matches the actual fault. If the suggested fault does not match the actual fault, represent that cause with False. If it does match the true fault, represent it with a True. For example, for an observation of 'the truck does not slow down when pressing the brake pedal' with actual fault of 'rear brakes do not apply' and recommended faults of {fault_1:'front breaks do not apply', fault_2:'rear brakes do not apply'}, you should represent it as {fault_1:False, fault_2:True}

In all RAG experiments, the PDF extraction package used is PyMuPDF (Rauber & Inc., 2024). Records (chunks or structured content by fault code) are represented by fixed-length vectors in the *all-MiniLM-L6-v2*'s learned latent space. A chunk size of 1000 characters is used. The technical content is organized in a local *txtai* vector database (Mezzetti, 2020) using the pre-trained *all-MiniLM-L6-v2* encoder model (Sentence Transformers, 2021), a down-sized implementation of MiniLM (Wang et al., 2020).

The next sections explore how observation data is used with the Fault Recommender System as a validation exercise for system prototyping, using both quantitative and qualitative assessment approaches. The outcomes of these experiments provide feedback which is used to inform system design tweaks. This case study will walk through two iterations of the process.

4. CHAT-BASED RECOMMENDER TASK

The first task for our system was designed to mimic a troubleshooting scenario using an out-of-the-box RAG-based chat application. Technical manuals were stored in a vector database for system retrieval, with users entering values via a chat interface. The system generated a list of potential faults from the observation.

The Recommender agent was configured in three RAG variations, compared against an unaugmented baseline ("LLM-only"). These variations explored different database storage models. The first used an out-of-the-box PDF extraction package to chunk text from the manual ("Rag 1: Raw & Chunked"). The second cleaned the data post-extraction before chunking ("Rag 2: Cleaned & Chunked"). The third utilized a structured schema with cleaned troubleshooting tasks indexed by fault code in a tabular format ("Rag 3: Structured").

Table 2. Results comparing the Chat-based Recommender Task across the different experiments.

Metric	LLM Only	RAG 1: Raw & Chunked	RAG 2: Cleaned & Chunked	RAG 3: Structured
Mean f1BOW	0.25	0.34	**0.36**	0.35
Median f1BOW	0.24	0.31	0.31	**0.32**
Mean BLEU	0.18	0.23	0.24	**0.25**
Median BLEU	0.17	0.20	0.20	**0.22**
Mean ROUGE	0.22	0.27	**0.28**	**0.28**
Median ROUGE	0.20	**0.25**	**0.25**	**0.25**
Hallucination Rate	94.4%	78.6%	78.3%	77.3%
Evaluator Hit Rate	99.7%	99.0%	99.0%	98.0%
Evaluator Highest Score Agreement	60.6%	61.8%	62.1%	63.5%
Latency (Per 100 Observations)	260 sec	290 sec	269 sec	257 sec

4.1. Quantitative Performance

The Chat task produced unstructured text. F1 BOW, BLEU, and ROUGE metrics compared recommendations against ground truth, with higher values indicating greater similarity. Table 2 and Figure 2 present the mean and median values, alongside the distribution of these highest values per observation. Generally, RAG configurations yielded closer matches compared to the "LLM-only" baseline.

Hallucination rate measures how many suggestions were present in the technical manual, which we calculated using the list of fault codes. Python's difflib.get_close_matches() function identified the closest corresponding manual entries by applying a cutoff threshold of 0.6. Suggestions matching this criterion were considered potential matches. The percentage of such suggestions determined the hallucination rate, which was notably high in this task due to matching responses only to a set of labels which were not supplied explicitly to the LLM. For this reason, we expect the hallucination rate to be very high in the "LLM-only" baseline as it had no manual or codes to infer off of, which they were with a value of 94%. Latency was calculated as the total time in seconds to run for 100 observations.

To look at how the Evaluator Agent scored the systems, we calculated the "Evaluator hit rate". Hit Rate is a metric for evaluating unranked responses from a recommendation system, measuring the fraction of observations where the true label is included in the suggested set. The Evaluator hit rate indicated a high number of observations with at least one "True" suggestion, which is more an indicator that the Evaluator nearly always picked one response as the most likely to match the ground truth, indicating need to calibrate the Evaluator Agent in this system. We also compared the Evaluator's scoring to the "Best Match" score, which was determined by summing the BLEU, ROUGE and F1 BOW metrics and selecting the highest combined score for each observation. Evaluator Highest Score Agreement is calculated as the percentage of recommendations where the scores agree. These numbers are consistently around two thirds of the time for all four experiments.

Table 3. Example of recommended possible faults for the Chat-based Recommender task. The top shows the input observation with the True Fault Code. Two experiments are shown: LLM-only on the left and RAG 3: Structured on the right.

Observed by Operator	a lot of water comes out when draining air tanks
(True) Fault Code	LARGE QUANTITY OF MOISTURE EXPELLED FROM AIR RESERVOIRS

	LLM-Only Recommended Possible Faults	RAG 3: Structure Recommended Possible Faults
1	Faulty or worn-out air tank drain valves	Large quantity of moisture expelled from air reservoirs
2	Excessive moisture buildup in the air tanks	Air system pressure builds up more than 120 PSI (827 KPA) (Compressor fails to unload)
3	Leaking air tank seals or connections	Air system loses pressure during operation/slow, no, or incorrect air pressure buildup
4	Clogged air tank drain lines	Air dryer purges continually
5	Malfunctioning air compressor	Excessive condensation in fuel

4.2. Qualitative Performance

Qualitative performance evaluation is important for understanding and interpreting model behavior. While looking at individual responses provides insight on model behaviors, it can also bias interpretation toward only the first few examples. To address this, we applied an error analysis ap-

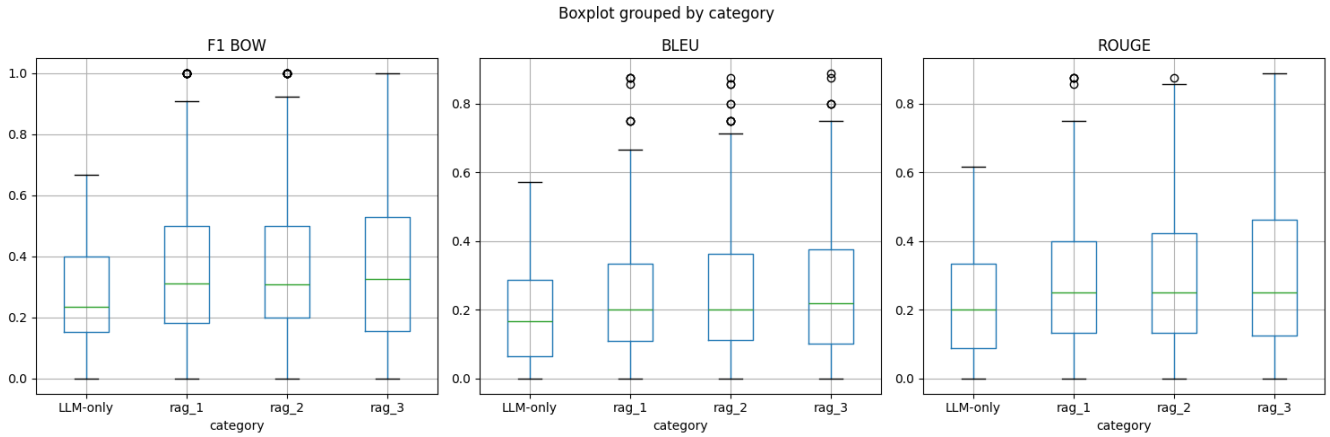

Figure 2. Boxplot comparing F1 BOW (left), BLEU (middle), and ROUGE (right) scores for the "Chat task" which compare the ground truth fault to the Recommender Agent suggestions. The highest scored suggestion for each observation is used. Values tend to be higher for the RAG experiments.

proach where we systematically reviewed model outputs and grouped them into categories of mistakes.

Early inspection revealed a recurring pattern: model responses often described a root cause rather than a fault code. Fault codes in the manual are defined in terms of operator-observable conditions ("as observed by operator"), while causes reflect underlying mechanisms of failure. An example is shown in Table 3. The "LLM-only" baseline generated recommendations such as "faulty or worn out air tank valves," which are possible causes of the observed fault, while the "RAG 3: Structured" case produced recommendations more resembling as operator observations, aligning more closely with the troubleshooting steps in the manual.

To systematically measure this behavior, we randomly sampled 20 observations from the 300 test cases and labeled each recommended fault as either "Operator Observation" or "Possible Cause." While the illustrative example highlights a case where RAG 3: Structured aligned with operator observations, in most cases recommendations across systems leaned toward possible causes. The distribution of these labels is summarized in Table 4.

This experiment could iterate in a few directions. One direction could be by setting the dataset to grab the initial troubleshooting step (or all steps) associated with the ground truth fault code and asking the system to return either the first step or a recommended ordered set of troubleshooting steps. Based on the observation that the system tended to suggest possible causes, we could alternatively modify the system to suggest possible causes for the observed fault and compare against root cause descriptions in the ground truth.

Another direction could be by providing the system with labels to select from. We chose this direction to explore next for a few reasons: first, the fault code was available and accessible and not being used. Second, it would enforce the level of information (operator observed fault code) in the generated recommendations. Third, scoring and evaluation should differ when systems have labels to select from, so we wanted to explore how that would change when the nature of the task changes.

Table 4. Qualitative Error Analysis. Percentage of 20 randomly sampled observations which were Operator Observations (as opposed to Possible Causes)

Case	**Operator Observation**	**%**
LLM Only	0	0%
RAG 1: Raw & Chunked	4	20%
RAG 2: Cleaned & Chunked	1	5%
RAG 3: Structured	4	20%

5. LABEL-CONSTRAINED FAULT CLASSIFICATION TASK

The Label-Constrained Fault Classification Task builds directly on the limitations observed in the Chat-based Recommender. Instead of free-form fault descriptions, we constrained outputs to a structured list of fault codes and required predictions to come from that list. This change alters the task from open-ended recommendations to label-matching classification, aligning outputs with the operator-observable fault codes already defined in the manuals.

Because recommendations now draw from a fixed label set, text-similarity metrics such as BLEU, ROUGE, and F1 BOW are less meaningful. Instead, performance is measured by

whether the system includes the ground truth code among its recommendations ("hit rate") and by the agreement between the ground truth and the Evaluator Agent's judgments.

We repeated the same four experimental conditions from the chat-based task, with one addition: a "Retrieval-only" baseline that directly returns fault codes from the manual without using an LLM. This baseline mirrors a simple retrieval-based fault detection approach and provides a useful reference point for comparing the added value of LLM integration.

5.1. Quantitative Performance

The results revealed that across all metrics, the Retrieval-only baseline outperformed every LLM-based approach, achieving an 86% hit rate compared to 70% for "LLM-only" and 57–64% for RAG variants (Table 5). The Evaluator Agent had lower hit rates than in the chat-based task, likely reflecting the added constraint of mapping outputs to specific labels. Still, Evaluator–ground truth agreement was roughly two-thirds across experiments (except "Retrieval-only"), similar to the chat-based setting.

One challenge is the large label space: 224 possible fault codes, many of which are very similar. The codes are organized into a two-level hierarchy of 22 systems (e.g. Electrical, Engine, Suspension and Air Cooling). When recommendations were evaluated at the system level, System Level Hit Rates rose dramatically, ranging from 87% (RAG 3: Structured) to 96% (Retrieval-only). This suggests that while models may struggle to pinpoint the exact fault code, they are more reliable at narrowing down to the right subsystem, a level of granularity that may be sufficient for guiding an operator toward the correct troubleshooting path.

5.2. Qualitative Performance

Although this task reduces to outputs to labels, contrasting error modes highlights where different approaches succeed or fail. Manual review suggested that Retrieval-only methods performed best when operator observations closely matched fault code phrasing (such as "cab does not raise" mapping to "Cab Will Not Raise"), reflecting strong surface-level similarity matching.

By contrast, we observed LLM-based systems occasionally succeeded where retrieval did not, particularly when abstraction or paraphrasing was required. For example (shown in Table 7), the observation "temperature gauge rises into the red zone during operation" corresponds to the code "Engine Overheats." Here, the LLM-based cases produced the correct label, while Retrieval-only suggested codes emphasizing "temperature" but missed the intended meaning.

To formalize these observations, we grouped cases into categories based on which systems were correct (Table 8). About 45% of cases were correct or mostly correct across the five experimental conditions, so error analysis should focus on the remaining 55% of cases. Notably, in roughly 17% of cases both Retrieval and LLMs succeeded while two or more RAG variants failed (labeled as "RAG adds noise" in the table), illustrating how retrieval augmentation can sometimes introduce noise. Viewed this way, the LLM-only approach can be interpreted as acting like a semantic similarity approach which matches the input description against the list of fault codes. From this perspective, LLM-only is functionally closer to retrieval than to open-ended generation.

From here, further analysis could examine how cases are distributed across these categories, both to explain the quantitative metrics and to support interpretability of the system. Manual review could identify linguistic or semantic patterns that drive groupings, while statistical methods such as feature importance or predictors of category membership could highlight systematic factors influencing success or failure, which could help guide modeling choices and refinements.

6. DISCUSSION

This study aimed to prototype fault recommendation systems using LLMs and retrieval, and to use evaluation results to inform design requirements. Through two case study tasks, we demonstrated evaluation approaches for both an open-ended chat task and a label-constrained task, showing how quantitative and qualitative methods can be combined to guide system development. Three lessons emerged: (1) while LLMs provide flexible reasoning and abstraction, their outputs must be calibrated against expected results aligned with use-case goals; (2) retrieval methods may be more effective for direct label-matching tasks; and (3) evaluation-first design exposes strengths and weaknesses early, enabling a process for requirement setting and refinement during system prototyping.

The findings are subject to several limitations. The synthetic dataset was narrow in scope and generated using LLMs, which can limit retrieval performance (Soboroff, 2025). Moreover, the open-source manual likely appeared in LLM training data, whereas proprietary manuals may produce different results. Integration of additional sources, such as historical maintenance data, may further enhance performance by prioritizing common faults and incorporating case-based reasoning. Refining the prompt, such as by providing examples to the Recommender Agent or incorporating a prompt-optimizer tool in the system, may also impact performance.

Evaluation methods were also limited. For recommender systems, metrics beyond unranked matching would provide a more complete view of performance. Ranked measures (e.g., HitRate@N) and human-in-the-loop evaluations could yield more realistic assessments. In addition, our Evaluator Agent was often overly permissive, reducing its descriminatory power. Improved Evaluator Agents should be calibrated and provide justifications such as rationales or certainty.

Table 5. Results comparing experiments for the Label-Constrained Fault Classification Task.

Metric	LLM Only	RAG 1: Raw & Chunked	RAG 2: Clean & Chunked	RAG 3: Structured	Retrieval Only
Hit Rate Ground Truth	69.6%	64.3%	62.3%	57.3%	**86.3%**
Hit Rate According to the Evaluator Agent	86.3%	88.7%	89.3%	84.7%	97.3%
Evaluator Ground Truth Agreement	69.2%	64.0%	62.0%	56.7%	**86.0%**
Accuracy	96.2%	93.6%	93.2%	93.2%	96.7%
Precision	79.3%	66.9%	65.0%	63.2%	84.0%
Recall	99.5%	99.5%	99.5%	98.8%	99.6%
f1 Score	88.3%	80.0%	78.6%	77.1%	91.2%
Hallucination Rate	0.14%	0.94%	1.41%	0.54%	0.00%
Latency (Per 100 Observations)	301 sec	323 sec	334 sec	334 sec	117 sec

Table 6. Hit rate comparison at the system level, which is a higher level than fault codes on the fault hierarchy.

Metric	LLM Only	RAG 1: Raw & Chunked	RAG 2: Clean & Chunked	RAG 3: Structured	Retrieval Only
System Level Hit Rate	87.0%	87.3%	91.3%	90.7%	**95.7%**
Evaluator System Level Hit Rate	93.6%	92.3%	90.7%	87.7%	99.0%

From the performance of a Fault Recommender system perspective, there is a balance between retrieval- and LLM-based approaches. Retrieval is effective for surface-similar observations, while LLMs may add value in more abstract or complex cases. Structured storage, such as knowledge graphs, may further improve retrieval, and hybrid approaches are a promising direction. For example, the HybridRAG approach combines traditional information retrieval, RAG, and knowledge graph retrieval for improved recommendations (Xie et al., 2025). However, hybrid designs must incorporate evaluation from the outset and account for trade-offs in latency, complexity, and resource usage.

6.1. Industrial Lessons Learned

The strengths of LLMs are their abilities for flexible reasoning (such as handling complex fault modes) and conversational interaction. Based on the lessons from this study, we recommend the following practices for developing LLM-based recommendation systems in industrial contexts:

Pick the tool to match the goals of the use case. When setting up the system, requirements should be clearly specified, including inputs, outputs, users, and success criteria. These requirements guide design choices, such as evaluation framing, data preparation, tools to use (such as LLMs), and computational constraints. For instance, in high-stakes applications that require efficient interaction with large volumes of technical content, retrieval-based approaches may be more suitable than LLM-based ones.

Adopt evaluation-first design, not demo-driven validation. Many AI-first systems are validated through polished demo dialogues that may provide convincing illustrations but not representative of overall system behavior. An evaluation-first approach resists such "demo-driven validation" in favor of systematic scoring. Evaluation-first design uncovers weaknesses early, enabling designers to refine the system based on observed behavior across a wide range of inputs. This process also helps build user trust and supports safety and compliance requirements.

Pilot and trial LLM systems under realistic conditions. If LLM-based approaches are deemed feasible and adopted, evaluation of the LLM-based system must account for flexibility, interactivity, and be capable of handling around an LLM's ability to synthesize complex information in dialogue. The dialogue layer introduces interactional and cognitive factors, is inherently open-ended, often lacking a single correct answer. Therefore, pilot studies are essential. Pilot studies, such as those described in (Löwhagen et al., 2025), are needed to evaluate LLM-based troubleshooting systems under realistic conditions.

The key industrial lesson is not that LLMs are always beneficial (in fact, our results show cases where they are not), but that evaluation-first design exposes potential issues upfront, supporting safer and more effective system development. Our direct takeaways are: (1) clearer guidance for prototyping as we digitize technical content to support maintenance recommender systems; (2) justification to begin defining requirements for a user trial pilot; and (3) as evidence for stakeholders eager to adopt LLMs, demonstrating that depending on requirements, other approaches may be the better path.

Table 7. Example of recommended possible faults for the Label-Constrained Fault Classification task for two experiments. The top shows the input observation with the True Fault Code.

Observed by Operator	temperature gauge rises into the red zone during operation
(True) Fault Code	ENGINE OVERHEATS

	LLM-Only Recommended Possible Faults	**Retrieval-only Recommended Possible Faults**
1	**ENGINE OVERHEATS**	HIGH ENGINE TEMPERATURE INDICATOR ILLUMINATES
2	OIL IN COOLING SYSTEM	WATER TEMP GAGE DOES NOT OPERATE OR IS INACCURATE.
3	LOSS OF COOLANT	HIGH ENGINE TEMPERATURE INDICATOR DOES NOT ILLUMINATE.
4	WATER TEMP GAGE DOES NOT ILLUMINATE	LOSS OF COOLANT
5	MASTER STOP INDICATOR DOES NOT ILLUMINATE	TRANSMISSION TEMPERATURE INDICATOR DOES NOT ILLUMINATE

Table 8. Summary of qualitative analysis for label-constrained fault classification task. Grouping observations can help in analysis of system behavior across the full dataset.

Case	**Observations Percentage**
Mostly Correct (Either All Correct, or Retrieval Correct & LLM-only Correct with 2 RAG Correct)	45.0%
Retrieval preferred (Retrieval Correct, Mix of RAG Correct or Incorrect, LLM-only Incorrect)	24.7%
RAG adds noise (Retrieval Correct, Most RAG Incorrect, LLM-only Correct)	16.7%
LLM potentially preferred (Retrieval Incorrect, Mixed Results)	10.0%
All Incorrect	3.7%

ACKNOWLEDGMENT

This work was financially supported by LMI Applied Research and Partnerships.

NOMENCLATURE

LLM	Large Language Model
RAG	Retrieval-Augmentation Generation
TLP	Technical Language Processing
AI	Artificial Intelligence
PHM	Prognostics and Health Management
CBR	Case-Based Reasoning
KG	Knowledge Graph
LMTV	Light Medium Tactical Vehicles
BOW	Bag of Words

REFERENCES

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... others (2023). GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Adhikari, N. S., & Agarwal, S. (2024). A comparative study of pdf parsing tools across diverse document categories. *arXiv preprint arXiv:2410.09871*.

Alghamdi, E., Halvey, M., & Nicol, E. (2024). System and user strategies to repair conversational breakdowns of spoken dialogue systems: a scoping review. In *Proceedings of the 6th ACM Conference on Conversational User Interfaces* (pp. 1–13).

Azevedo, N., Aquino, G., Nascimento, L., Camelo, L., Figueira, T., Oliveira, J., ... Figueiredo, C. (2023). A novel methodology for developing troubleshooting chatbots applied to atm technical maintenance support. *Applied Sciences*, *13*(11), 6777.

Chen, A., Tian, Y., Zhang, J., Li, C., & Zhang, H. (2025). LLM-based intelligent Q&A system for railway locomotive maintenance standardization. *Scientific Reports*, *15*(1), 12953.

Dave, A. J., Nguyen, T. N., & Vilim, R. B. (2024). Integrating llms for explainable fault diagnosis in complex systems. *arXiv preprint arXiv:2402.06695*.

Doris, A. C., Grandi, D., Tomich, R., Alam, M. F., Cheong, H., & Ahmed, F. (2024). DesignQA: A Multimodal Benchmark for Evaluating Large Language Models' Understanding of Engineering Documentation. *arXiv preprint arXiv:2404.07917*.

Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., ... Larson, J. (2024). From Local to Global: A Graph RAG Approach to Query- Focused Summarization. *arXiv preprint arXiv:2404.16130*.

Eleti, H. J., Atty, & Kilpatrick, L. (2023, June). *Function Calling and Other API Updates.* `https://openai.com/index/function-calling-and-other-api-updates/`. (Accessed: 2024-06-25)

Ferdousi, R., Hossain, M. A., Yang, C., & Saddik, A. E. (2024). Defecttwin: When llm meets digital twin for railway defect inspection. *arXiv preprint arXiv:2409.06725*.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

He, H., Huang, J., Li, Q., Wang, X., Zhang, F., Yang, K., ... Chu, F. (2024). Maintagt: Sim2real-guided multimodal large model for intelligent maintenance with chain-of-thought reasoning. *arXiv preprint arXiv:2412.00481*.

Hodkiewicz, M., Klüwer, J. W., Woods, C., Smoker, T., & Low, E. (2021). An ontology for reasoning over engineering textual data stored in fmea spreadsheet tables. *Computers in Industry*, *131*, 103496. Retrieved from https://www.sciencedirect.com/science/article/pii/S0166361521001032 doi: https://doi.org/10.1016/j.compind.2021.103496

Huang, H., Shah, T., Karigiannis, J., & Evans, S. (2024). Physics and data collaborative root cause analysis: Integrating pretrained large language models and data-driven ai for trustworthy asset health management. In *Annual Conference of the PHM Society* (Vol. 16).

Jadon, A., Patil, A., & Kumar, S. (2025). Enhancing domain-specific retrieval-augmented generation: Synthetic data generation and evaluation using reasoning models. *arXiv preprint arXiv:2502.15854*.

Karray, M. H., Ameri, F., Hodkiewicz, M., & Louge, T. (2019). ROMAIN: Towards a BFO compliant reference ontology for industrial maintenance. *Applied Ontology*, *14*(2), 155–177.

Khan, A., Nahar, R., Chen, H., Flores, G. E. C., & Li, C. (2025). Faultexplainer: Leveraging large language models for interpretable fault detection and diagnosis. *Computers & Chemical Engineering*, 109152.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... others (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, *33*, 9459–9474.

Liang, Z., Xu, Y., Hong, Y., Shang, P., Wang, Q., Fu, Q., & Liu, K. (2024). A survey of multimodel large language models. In *Proceedings of the 3rd International Conference on Computer, Artificial Intelligence and Control Engineering* (pp. 405–409).

Löwenmark, K. (2025). *Technical Language Supervision and Agentic AI for Condition Monitoring* (Unpublished doctoral dissertation). Luleå University of Technology.

Löwenmark, K., Strömbergsson, D., Liu, C., Liwicki, M., & Sandin, F. (2025). Agent-based condition monitoring assistance with multimodal industrial database retrieval augmented generation. *arXiv preprint arXiv:2506.09247*.

Löwenmark, K., Taal, C., Schnabel, S., Liwicki, M., & Sandin, F. (2021). Technical language supervision for intelligent fault diagnosis in process industry. *arXiv preprint arXiv:2112.07356*.

Löwhagen, N., Schwendener, P., & Netland, T. (2025). Can a troubleshooting ai assistant improve task performance in industrial contexts? *International Journal of Production Research*, 1–22.

Lu, W., Luu, R. K., & Buehler, M. J. (2025). Fine-tuning large language models for domain adaptation: Exploration of training strategies, scaling, model merging and synergistic capabilities. *npj Computational Materials*, *11*(1), 84.

Lukens, S., McCabe, L. H., Gen, J., & Ali, A. (2024). Large Language Model agents as prognostics and health management copilots. In *Proceedings of the Annual Conference of the PHM Society* (Vol. 15).

Meng, X., Jing, B., Wang, S., Pan, J., Huang, Y., & Jiao, X. (2023). Fault knowledge graph construction and platform development for aircraft PHM. *Sensors*, *24*(1), 231.

Mezzetti, D. (2020). *txtai: the all-in-one embeddings database.* Retrieved from https://github.com/neuml/txtai

Peng, H., & Yang, W. (2024). Knowledge graph construction method for commercial aircraft fault diagnosis based on logic diagram model. *Aerospace*, *11*(9), 773.

Qaid, H. A., Zhang, B., Li, D., Ng, S.-K., & Li, W. (2024). Fd-llm: Large language model for fault diagnosis of machines. *arXiv preprint arXiv:2412.01218*.

Rajpathak, D. G. (2013). An ontology based text mining system for knowledge discovery from the diagnosis data in the automotive domain. *Computers in Industry*, *64*(5), 565-580. Retrieved from https://www.sciencedirect.com/science/article/pii/S0166361513000456 doi: https://doi.org/10.1016/j.compind.2013.03.001

Rauber, J. X., & Inc., A. S. (2024). *PyMuPDF - Python bindings for MuPDF.* https://pymupdf.readthedocs.io/. (Version 1.23.25)

Schaafstal, A., Schraagen, J. M., & Van Berl, M. (2000). Cognitive task analysis and innovation of training: The case of structured troubleshooting. *Human factors*, *42*(1), 75–86.

Sentence Transformers. (2021). *all-MiniLM-L6-v2: Sentence transformers model.* https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

Shin, H., Tien, K.-W., & Prabhu, V. (2019). Modeling the maintenance time considering the experience of the technicians. In *IFIP International Conference on Advances in Production Management Systems* (pp. 716–721).

Soboroff, I. (2025). Don't use LLMs to make relevance judgments. *Information retrieval research journal*, *1*(1), 10–54195.

Tang, X., Chi, G., Cui, L., Ip, A. W., Yung, K. L., & Xie, X. (2023). Exploring research on the construction and application of knowledge graphs for aircraft fault diagnosis. *Sensors*, *23*(11), 5295.

Tao, L., Liu, H., Ning, G., Cao, W., Huang, B., & Lu, C.

(2025). Llm-based framework for bearing fault diagnosis. *Mechanical Systems and Signal Processing*, *224*, 112127.

Trilla, A., Yiboe, O., Mijatovic, N., & Vitri à, J. (2024). Industrial-grade smart troubleshooting through causal technical language processing: a proof of concept. *arXiv preprint arXiv:2407.20700*.

U.S. Department of the Army. (1998, June). Technical Manual TM 9-2320-365-10: M1078 Series Operators Manual [Computer software manual]. Retrieved 2025-06-21, from https://www.steelsoldiers.com/upload/M1078/m1078_TM%209-2320-365-10.pdf (Accessed via SteelSoldiers.com)

U.S. Department of the Army. (2008, July). *MIL-STD-3031: Preparation of Digital Technical Information for Equipment Maintenance.* https://quicksearch.dla.mil/. (Military Standard, Department of Defense, United States)

Vachtsevanos, G. J., Lewis, F., Roemer, M., Hess, A., & Wu, B. (2006). *Intelligent fault diagnosis and prognosis for engineering systems* (Vol. 456). Wiley Online Library.

Vidyaratne, L., Lee, X. Y., Kumar, A., Watanabe, T., Farahat, A., & Gupta, C. (2024). Generating troubleshooting trees for industrial equipment using large language models (llm). In *2024 ieee international conference on prognostics and health management (icphm)* (pp. 116–125).

Vitale, M., Youssef, A., Mishra, P., Shetty, S., Sharma, M., Vanzo, G., ... Bettini, A. (2024). Harnessing generative ai for interactive system failure diagnostics: A user-centric approach to streamlined problem solving and maintenance. In *Abu Dhabi International Petroleum Exhibition and Conference* (p. D011S020R006).

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. (2020). MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. *Advances in Neural Information Processing Systems*, *33*, 5776–5788.

Woods, C., French, T., Hodkiewicz, M., & Bikaun, T. (2023). An ontology for maintenance procedure documentation. *Applied Ontology*, 1–38.

Woods, C., Selway, M., Bikaun, T., Stumptner, M., & Hodkiewicz, M. (2023). An ontology for maintenance activities and its application to data quality. *Applied Ontology*, 1–34. doi: 10.3233/SW-233299

Xie, X., Tang, X., Gu, S., & Cui, L. (2025). An intelligent guided troubleshooting method for aircraft based on hybirdrag. *Scientific Reports*, *15*(1), 17752.

Zheng, S., Pan, K., Liu, J., & Chen, Y. (2024). Empirical study on fine-tuning pre-trained large language models for fault diagnosis of complex systems. *Reliability Engineering & System Safety*, *252*, 110382.

Zhu, K., Luo, Y., Xu, D., Yan, Y., Liu, Z., Yu, S., ... others (2024). RAGEval: Scenario specific rag evaluation dataset generation framework. *arXiv preprint arXiv:2408.01262*.

BIOGRAPHIES

Sarah Lukens is a Data Science Fellow at LMI. Her interests are focused on data-driven modeling for reliability applications by combining modern data science techniques with current industry performance data. This work involves analyzing asset maintenance data and creating statistical models that support asset performance management (APM) work processes using components from natural language processing, machine learning, and reliability engineering. Sarah completed her Ph.D. in mathematics in 2010 from Tulane University with focus on scientific computing and numerical analysis. Sarah is a Fellow of the PHM Society and a Certified Maintenance and Reliability Professional (CMRP).

Matthew Bishof is a Senior Consultant in Logistics Engineering at LMI. His work has primarily focused on digital engineering and the modernization and utilization of manufacturing and sustainment data. Recently, he has been the product lead for maintenance solutions leveraging artificial intelligence and legacy technical documentation. Matthew received a Bachelors of Science in Industrial and Systems Engineering in 2020 from Virginia Polytechnic Institute and State University.

Nadir Siddiqui is a Senior Analyst in AI and Machine Learning at LMI. His focus is on deploying AI solutions in government environments, identifying opportunities where applied AI can improve efficiency and support responsible decision-making. Nadir holds a Masters of Science in Data Science from the University of Virginia.

Destiny West is a Project, Program, and Operations Management Professional at LMI. With 21 years of military experience in the aerospace and defense industry, Destiny specializes in maintenance operations and fleet health management for advanced aircraft systems such as the F-15, F-16, HH-60, F-22, F-35, and U-2. She has led cross-functional teams in high-pressure environments, overseeing aircraft sustainment, developing maintenance schedules, and managing performance metrics. Her roles have involved providing strategic guidance on maintenance plans, ensuring compliance with directives, and enhancing mission readiness through detailed oversight. Destiny holds a Bachelor's degree in Business Administration and Management from Saint Leo University and has advanced expertise in maintenance production management from the Community College of the Air Force.