DrAgent: An Agentic Approach to Fault Analysis in Power Grids Using Large Language Models

Barun Kumar Saha
Grid Automation R&D
Hitachi Energy
Bangalore 560048, India
barun.kumarsaha@hitachienergy.com

Aarthi V
Hitachi Energy Research
Hitachi Energy
Bangalore 560048, India
aarthi.v@hitachienergy.com

O. D. Naidu
Hitachi Energy Research
Hitachi Energy
Bangalore 560048, India
Od.Naidu@hitachienergy.com

***Abstract*—Power grids are often subjected to faults. In response, Disturbance Records (DRs) are generated by relevant devices, capturing a brief snapshot of the pre-fault and post-fault conditions. Therefore, DR analysis is a critical requirement for fault diagnosis, which, however, requires both expert knowledge and time. In this work, we investigate the use of Large Language Models (LLMs) to address these challenges. In particular, we design DrAgent, an agentic workflow aimed to make DR analysis potentially simpler and more accessible. DrAgent combines a pre-trained LLM with user-defined, domain-specific tools, allowing it to achieve relatively higher accuracy in relevant cases. Based on an input query, the agent uses an LLM to plan a sequence of steps to address the query, involving one or more tool usage. In order to improve the robustness, we use argument reconstruction to try and resolve potential syntactic errors in the planned function calls. Moreover, we cache the outputs of frequently used tools, in order to reduce redundancy and latency to an extent. We evaluated the performance of DrAgent using real-life DRs and a set of evaluation questions. The answers generated by DrAgent were assigned a numerical score by a domain expert. The results of performance evaluation indicate that DrAgent, in general, can significantly improve the accuracy of responses requiring numerical results. On the other hand, the step-by-step reasoning of agentic workflow can potentially help power grid operators, both novice and expert, to learn or verify the process.**

***Index Terms*—Power Grids, Disturbance Records, COMTRADE, Artificial Intelligence, Large Language Models, Agents, Function Calling**

I. Introduction

Power grids are critical infrastructure, acting as the lifeline of modern industry and society. However, they are often subjected to faults due to various reasons, such as natural hazards and electromechanical failures. In response to faults, Intelligent Electronic Devices (IEDs) generate DRs, typically following the IEEE Common Format for Transient Data Exchange (COMTRADE) [1] standard. Fast identification and remedy of the faults is necessary since every minute of transmission line outage can lead to a high monetary penalty [2].

A DR captures a brief snapshot of the operating conditions of the segment of a power grid connected to an IED. Analysis of DRs can help to diagnose a fault, which, in turn, helps address other problems, such as estimating the distance of a fault along a transmission line [3]. However, DR analysis requires expertise in power grids and power systems, in general. In other words, fault analysis can be a challenging task for the non-experts. On the other hand, in some complex scenarios, even the experts may require several minutes to hours to perform a detailed diagnosis of a fault. Moreover, as senior experts retire from service and new employees join the workforce, a potential knowledge gap is created where the existing expert knowledge, in part, might be lost. Accordingly, artificial intelligence (AI) appears to be a suitable candidate for addressing these challenges.

The recent advances in Generative AI (GenAI) and LLMs have influenced different aspects of contemporary work and life [4]. LLMs, which are pre-trained with vast corpora of text, can answer diverse queries based on their existing knowledge. However, at times LLMs may generate incorrect answers, often referred to as "hallucination." Moreover, while pre-trained LLMs typically have a "general" knowledge about the world, they often lack in having specialized, domain-specific knowledge. In addition, LLMs normally lack the ability to interact with the real world. Autonomous agents and agentic workflows [5]–[7] have emerged as a potential solution to address some of these challenges. In general, an agent uses its existing knowledge together with tools (or functions) to solve a given problem. Consequently, agentic workflows allow for an efficient representation of domain-specific knowledge for LLMs.

Motivated by these aspects, in this work, we investigate the design of DrAgent, an agentic workflow aimed to help with fault analysis in power grids. DrAgent, illustrated in Figure 1, is based on a popular agentic architecture, Reasoning Without Observation (ReWOO) [6]. In particular, given a DR and a query (alternatively, question or task), the agent formulates a plan to solve the query using one or more user-defined tools. Subsequently, the steps are executed sequentially by invoking relevant functions with appropriate parameters. Since LLM-generated function calls, as part of the plan to solve a task, sometimes may contain incorrectly formatted parameters, DrAgent uses argument reconstruction, where an LLM is asked to generate a syntactically correct function call.

On the other hand, in some cases, a given tool may be used to execute the same code for different tasks, leading to redundancy and overhead. DrAgent addresses this challenge

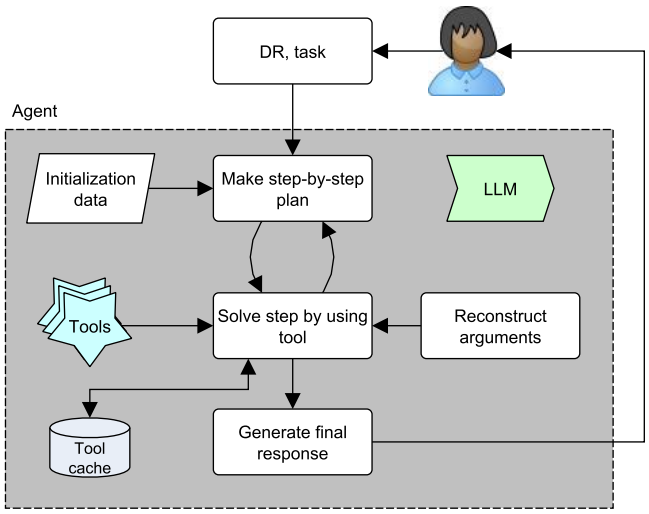

Fig. 1: Illustration of DrAgent, based on the ReWOO [6] architecture.

by caching the output of frequent function calls. Finally, after a query has been solved by executing all the designated steps, DrAgent provides the user with an appropriate response, consisting of text and/or image(s). When relevant, DrAgent also provides a brief explanation and reasoning behind the answer, potentially enabling users to understand and verify the fault analysis. The scope of DrAgent is limited to the analysis of faults occurring in the transmission lines of power grids.

The specific contributions of this work are as follows:

- Designing DrAgent, an agentic workflow based on ReWOO, to enable accurate fault and disturbance analysis in power grids by leveraging domain-specific knowledge.
- Improving the robustness of plan execution and response latency of agentic workflow, in part, by introducing argument reconstruction of function calls and tool output caching.
- Evaluating the accuracy of DrAgent's response to a set of frequently asked questions in fault analysis using DRs generated by IEDs. The evaluation was performed by a domain expert.
- Quantitatively comparing the performance of DrAgent (i.e., agentic workflow) and a non-agentic approach to fault analysis.

The remainder of this work is organized as follows: Section II presents a short background on fault analysis and agentic workflows. Section III discusses the design of DrAgent. Section IV presents the experimental setup. The results of the performance evaluation are discussed in Section V. Finally, Section VI concludes this work.

II. Background

AI has found widespread applications in the context of fault analysis and other areas in power grids and power systems. Saha and Parapurath [8], for example, investigated the problem of distinguishing between "real" and "test" DRs by clustering the waveform images based on their similarity. Real-life substations may involve manual operations and test data generation, so this solution constitutes a data-cleaning step. Hong et al. [9] identified the fault types by training a Deep Neural Network-based classification model using the images of the waveforms.

Chen et al. [10] surveyed the contemporary use of AI and machine learning toward solving different power systems-related problems, such as fault prediction. The authors observed that the integration of LLMs and tools may potentially help experts investigate domain-specific use cases. Gitzel et al. [11] outlined a vision of how LLMs can potentially help distribution system operators obtain tangible recommendations across various use cases. Xiao and Xu [12], on the other hand, investigated the problem of energy consumption optimization in buildings by leveraging LLM-based agents to interpret unstructured data using domain knowledge.

Reasoning and Acting (ReAct) [7] largely popularized the growth of LLM-based agents. ReAct employs a thought-action-observation loop, where an LLM, given a query, runs a tool and generates some output, and then decides on the next step based on the currently observed output. ReAct, therefore, requires continuous interaction with an LLM. Moreover, any wrong action can deviate the agent's trajectory from the desired path. ReWOO [6] addresses these challenges by making an LLM generate a sequence of plans (or steps) and evidence, based on the available tools. Moreover, the evidence (output) of each step is reused by one or more later steps. Consequently, a ReWOO agent can execute these steps by using tools locally. LLMCompiler [13], on the other hand, offers further improvement by enabling tool execution in parallel, thereby potentially reducing the response time.

To synthesize, the use of GenAI and LLMs for fault analysis in power grids largely remains unexplored. Since this is a recurring problem for grid operators, requiring both expertise and time, the investigation of an agentic approach gains importance.

On the other hand, in the context of LLM-based agents, it may be noted that, given the often unpredictable nature of LLMs, the syntax of invoking a tool, as prescribed by an LLM, might exhibit deviations from the correct syntax. As a consequence, such tool calls may fail. This can especially be true when the signatures of the underlying functions are non-trivial. As a motivating example, in a different context, Saha et al. [14] considered the use of domain-specific corrections to improve the accuracy of natural language text-to-SQL translation.

Moreover, in the course of a detailed analysis of any problem using an agent, certain tools may be invoked repeatedly using the same arguments, leading to redundant computations. In order to address these challenges and lacunae, in this paper, we take an agentic approach toward power grid fault analysis using LLMs, together with improving the robustness and non-redundancy of tool execution.

III. Design of DrAgent

DrAgent[1] is based on the ReWOO architecture. Therefore, it solves a query by generating a plan and executing the planned steps. In addition, the initialization data in Figure 1 pre-computes some of these data and makes them available to the LLM during planning. This avoids the need for an extra step (i.e., a function call) to retrieve the parameter's value, leading to simpler plans and lower response latency. In this section, we present a detailed overview of DrAgent.

A. System Model

Let $\mathcal{F}$ be the set of user-defined tools (functions) available to an agent. Let q be a given query. Let $\mathcal{P}(q) = \langle e_1 = f_1(a_1), e_2 = f_2(a_2), \cdots, e_k = f_k(a_k)\rangle$ be the k-step plan generated by the agent to address the query, where a_k denotes the set of arguments taken by the kth function call, generating output ("evidence") e_k; $f_i \in \mathcal{F}$. The set a_i includes one or more values from $e_1, e_2, \cdots, e_{i-1}$. In addition, $a_i, \forall i$, may also contain zero or more default or previously saved data.

Let κ be a counter that keeps track of how many times a function has been called. Let τ be the tool cache so that:

$$\tau(f, a) = \begin{cases} f(a) & \text{if } \kappa[(f, a)] < \theta \\ e^{f(a)} & \text{otherwise,} \end{cases} \quad (1)$$

where θ is a threshold; $e^{f(a)}$ denotes the cached output of the function call $f(a)$. The output of frequent function calls can be cached by appropriately setting the value of θ in (1).

It may be noted that caching should be used only with those tools where the source data do not change. For example, the content of a DR never changes, so the results of relevant computations based on a DR can be cached. On the other hand, a function, for example, get_current_weather, where the underlying data changes over time, should not be cached. Equation (1) can be modified to achieve this by maintaining a list of cache candidates.

It may be further noted that tool caching is different from LLM caching, a feature enabled by many contemporary LLMs and LLM frameworks. Therefore, one can use both techniques, either of them, or none at all. LLM caching, where a query-response pair is cached, can significantly decrease the response latency. However, this also leads to a potential problem. If an LLM generated a wrong answer and cached it, a user would continue to get the wrong answer when the same question is asked again. With tool caching, this impact is largely reduced. For example, certain tools, operating on DR data, generate deterministic results, making them suitable for caching. This allows an LLM to interpret the results appropriately—even if the current interpretation is wrong, the LLM still has a positive likelihood of offering a correct interpretation when asked again.

Tool caching is also different from Least Recently Used (LRU) caching since the former caches content based on a usage threshold. Therefore, in theory, users could specify what to cache and when. However, LRU can still be used in some cases. For example, when analyzing a DR, most operations need to read DR data to perform computations. The disk reading time can be potentially reduced by caching the content of a DR using LRU.

With an agentic approach, the name of any function f and its arguments set a is generated by an LLM based on $\mathcal{F}$, which can be large. When a is relatively large or complex, there is a chance that an LLM may fail to generate a faithfully based on the signature of f. Such syntax errors may be trivial, for example, a missing closing quote of a string. However, the presence of any syntax error ensures that the corresponding function call will fail.

Let ρ be the argument reconstruction function, which generates a correct function call by fixing any syntax error in the arguments; $\rho \notin \mathcal{F}$. To illustrate, let us consider a hypothetical Python function: `def display_text(text: str) -> str`. A call with syntax errors, for example, `display_text(text='Hello, world)`, would fail. However, by applying the argument reconstruction function, we get ρ(`display_text(text='Hello, world))` = `display_text(text='Hello, world')`, which can be executed successfully.

Ideally, ρ can be realized in the local environment of an agent by applying pattern matching, for example, based on regular expressions. However, when the arguments contain multiple parameters of different types, such pattern matching could become difficult. In addition, one or more such arguments may contain spaces, which can make the parsing more difficult. For example, the tool responsible for searching product manuals—discussed later in this section—takes a query as input. The query may contain several words and symbols separated by spaces or line separators.

For the sake of simplicity, we realize ρ by making an additional call to the LLM. In particular, the prompt to the LLM contains a function's signature as well as the generated arguments set, based on which a correct set of arguments is asked to be generated. We ask the LLM to return the arguments as a Python dictionary, which can be unpacked and used to call the corresponding function. As an optional, final step, we also used json_repair [15] to fix any remaining syntax error with the function call arguments. This was done only when parsing failed after argument reconstruction.

Algorithm 1 illustrates the steps of tool invocation with DrAgent. Let n_k be the maximum number of arguments in any arguments set a_i. Then, serialization of a_i is done in $O(n_k)$ time. The concatenation of two strings (line 2) to generate a string of length n takes $O(n)$ time; $n > n_k$. Moreover, let $O(f)$ be the asymptotic time complexity of any function f. Accordingly, the time complexity of Algorithm 1 becomes $O(\max\{O(n), O(f)\})$. Since most functions in

[1]We prefer an agentic approach to fully autonomous agents—where LLMs can generate and execute relevant code—due to different reasons. On one hand, LLM-generated code at times could be incorrect, especially when domain-specific, specialized calculations are considered. On the other hand, for security reasons, an additional step of verifying LLM-generated code would be required before executing such code. With user-defined tools, we make use of trusted code that can capture relevant logic appropriately.

Algorithm 1: Tool invocation by DrAgent

Input:
- f: The name of the function
- a: The arguments set of the function
- κ: Function call counter
- θ: Tool caching threshold
- ρ: Argument reconstruction function

Output: r: The output of function call

```
s ← Serialize a into a string
h ← Generate the hash of f||s
if κ[h] ≥ θ then
    r ← Cached output from τ(f, a)
else
    r ← ρ(f(a))
    if κ[h] ≥ θ then
        Cache the output r in memory
κ[h] ← κ[h] + 1
```

DrAgent iterate over the sample values in a DR, the string length, n, in general, is dominated by the time complexity of the functions. In other words, Algorithm 1 works in $O(f)$ time, which is the expected behavior.

B. Overview of Tools

Figure 2 illustrates some of the current and voltage waveforms from a sample DR along with some key events, such as fault inception and clearance. Fault analysis involves identifying these pieces of information, among others, by analyzing a DR.

We implemented a dozen of Python functions and equipped DrAgent with them. The tools allowed us to achieve primarily three functionalities—analysis of the faults, visualization of the waveforms, and answering queries based on product manuals. Figure 3 shows the distribution of the parameter count of these functions (tools). Most functions have 1–2 parameters, for example, the paths to the COMTRADE configuration and data files. A few functions have relatively more parameters. For example, the function to plot the analog waveforms takes channel numbers and sample numbers as optional inputs, allowing users to zoom in on specific regions. It may be noted that the LLM itself is also another (default) tool available to the agent.

The tools for fault analysis appropriately encapsulated domain-specific knowledge and performed computations based on data available in the DRs. In particular, a DR contains a configuration and a data file, among others. The DR configuration file contains a list of configured channels (analog and digital) along with some other settings. The data file, on the other hand, contains raw sample value measurements before and after the fault trigger. For example, if the sampling frequency is 1000, then a one-second window captured in a DR data file will contain 1000 samples, where each sample captures the current and voltage measurements for the analog

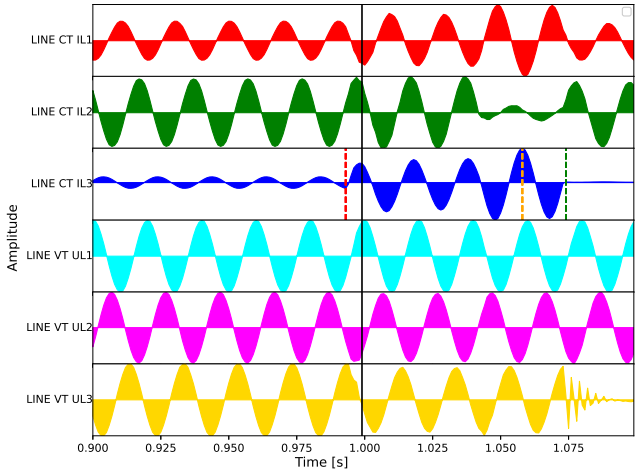

Fig. 2: Illustration of some key events based on a sample DR. This is a Phase 3 fault, as observed from the significantly distorted current waveform in the third row (Line CT IL3). The three dashed vertical lines, respectively, indicate the fault inception, the point when the maximum fault current was reached, and the fault clearance. The dark vertical line across the image indicates the trigger point.

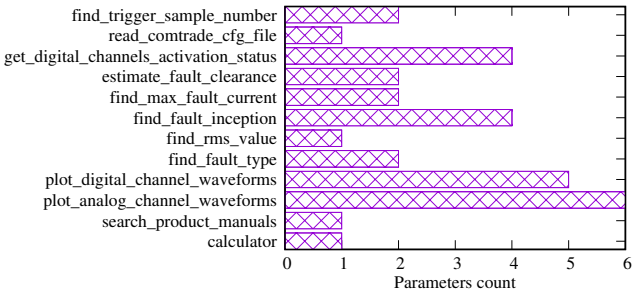

Fig. 3: Distribution of parameters count (x-axis) of the user-defined tools functions (y-axis).

channels as well as the active/inactive status for the digital channels.

In order to identify the fault type, for example, we prompted the LLM with a summary of current and voltage values before and after the trigger point, based on the DR data file. On the other hand, the fault inception was determined by identifying the sample number at which the current and voltage levels exceeded the normal measurements. These algorithms are commonly known and used by experts performing fault analysis. Therefore, the detailed steps are skipped for the sake of brevity.

To search across product manuals, we took a Retrieval-Augmented Generation (RAG) approach, via the search_product_manuals tool. In RAG [4], documents are typically split into smaller chunks, based on which vector embeddings are generated. These embeddings are stored in a vector database. Subsequently, given a query, semantically similar chunks are retrieved from the database, for example, based on cosine similarity between the embeddings of a

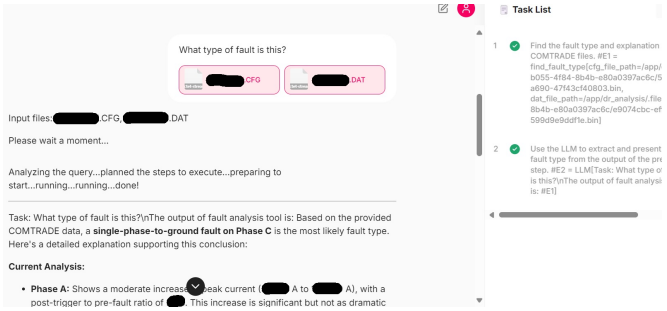

Fig. 4: A screenshot of the DrAgent prototype, depicting a query, the functions called, and a part of the answer. Some information has been redacted.

query and stored chunks. The list of chunks so retrieved is sent to an LLM based on which a response is generated. In many scenarios, the retrieved chunks are re-ranked based on contextual relevance before generating a response.

In our experiments, we observed that sometimes the LLM generates plans with nested function calls. Parsing and executing such plans lead to relatively more complex, recursive patterns. Therefore, we instruct the LLM to take a bottom-up approach to planning where (1) each step makes only one function call and (2) the pre-requisite function calls are planned to be executed ahead of the steps that need any such outputs as inputs. Such simplification allowed us to avoid dependency-related errors.

IV. Performance Evaluation

We implemented DrAgent using Python 3.12 and relevant libraries, such as Chainlit[2], comtrade[3], LangChain[4], and LangGraph[5]. We used the Gemini 1.5 Pro and Gemini 1.5 Flash [16] LLMs. The agent generated plans to solve tasks using Gemini 1.5 Pro. The application was containerized and deployed on the cloud. A container used 1 GB RAM and 1 CPU to run the application.

We used Vertex AI Agent Builder[6] to search across 138 product manuals using RAG. The search_product_manuals tool invoked relevant Application Programming Interfaces provided by the Agent Builder platform. The search results were re-ranked using FlashRank [17] and TinyBERT[7].

We evaluated the performance of DrAgent using 18 DRs generated by different IEDs. We considered six fundamental queries related to fault analysis, as shown in Table I. Each DR was manually analyzed by an expert, based on which the answers to these questions were separately documented.

For each DR, we asked DrAgent each of the six questions in sequence. The responses generated by DrAgent were captured

[2]https://docs.chainlit.io/get-started/overview
[3]https://github.com/dparrini/python-comtrade
[4]https://www.langchain.com/
[5]https://langchain-ai.github.io/langgraph/
[6]https://cloud.google.com/products/agent-builder
[7]https://huggingface.co/cross-encoder/ms-marco-TinyBERT-L-2

TABLE I: Questions used, and their relative importance, for the performance evaluation of DrAgent

#	Question	Importance
Q1	What is the type of the fault?	High
Q2	What is the fault inception?	High
Q3	What relay algorithm(s) is/are detected?	Medium
Q4	What is the trip output? Distance protection, differential protection, or both? Which phase?	Medium
Q5	What is the max fault current?	Low
Q6	When was the fault cleared?	Low

TABLE II: Evaluation Scale

Answer Type	Score
Incorrect answer	-1
No answer	0
Partially correct answer	1
Correct answer	2
Exceptionally well answered or provides valuable extra info	3

and saved in separate documents, one for each DR. Subsequently, these responses were manually evaluated by a domain expert. A numerical score was assigned to each answer.

We used a five-point scale to rate the accuracy of answers generated by DrAgent, as shown in Table II. Incorrect answers were assigned a score of -1. Sometimes the agent (or LLM) failed to generate an answer or claimed that it did not have an answer. In such scenarios, the score was 0. In some other scenarios, the answers generated were incomplete or partially correct. For example, if the agent, when asked about the fault type, responded with "single line-to-ground fault," but failed to specify which phase—A, B, or C—was involved, we considered the answer as incomplete and assigned a score of 1. Correct and complete answers were assigned a score of 2. Finally, a score of 3 was reserved for exceptionally good answers. Based on the scores for each DR and each question, the average score and the 95% confidence intervals were computed. In addition to the aforementioned six questions, we also executed other relevant queries to verify that appropriate function calls were made, in general.

In order to evaluate the potential impact of tool caching, we considered three additional queries, as shown in Table III. With the tool caching threshold (θ) set to 2, Q7, Q8, and Q9 were executed sequentially. This particular sequence made the latter two queries potentially use previously cached outputs of function calls. We considered the tool usage time—the time taken to execute the code contained in a tool, as given by (1)—to evaluate the potential impact of tool caching.

As a baseline, we also considered a non-agentic approach to fault analysis using Gemini 1.5 Pro and its multimodal capability. The objective was to identify whether or not the availability of user-defined, domain-specific tools can improve the accuracy of fault analysis.

In particular, given a DR, we prompted Gemini 1.5 Pro with a single prompt containing the six questions (Table I). In addition, the content of the DR configuration file as well as a portion of sample values from the DR data file were also provided. As mentioned earlier, the find_fault_type tool of DrAgent used a prompt that summarized the current and

TABLE III: Additional queries used for evaluating the performance of tool caching

#	Question
Q7	How long did the fault last?
Q8	Plot the analog waveforms. Show only 100 samples around the fault inception.
Q9	Plot the analog waveforms. Show only 50 samples around the fault clearance.

voltage levels. The same prompt was used with the non-agentic approach as well. Finally, images depicting the waveforms and status of the analog and digital channels, respectively, were also provided as part of the prompt used with the non-agentic approach.

Our prompt instructed Gemini to answer Q2 based on the raw sample value measurements from the DR data file, captured inside a small window around the trigger point identified by the DRs. The motivation was that a fault usually occurs around the trigger point. For Q5, we added instructions to identify the maximum current based on the peaks of the current waveforms observed in the image. Finally, for Q6, we instructed Gemini to identify the fault clearance based on a separate image showing the current and voltage waveforms after the trigger point.

In this context, it may be noted that the numerical answers require a high degree of accuracy. Otherwise, they are deemed as incorrect (score = -1). For example, fault inception and clearance (time measurements) may have a maximum absolute error of up to 3 milliseconds. On the other hand, for fault current measurement, we considered a tolerance range of 5%.

V. Results

Table IV shows the evaluation scores of DrAgent for each DR and each question, based on the evaluation performed by an expert. For DR13 through DR18, Q4 could not be answered. Accordingly, their scores are indicated as "N/A."

Table IV shows that the average score for eight DRs (rightmost column) was at 2.00, indicating that DrAgent was able to generate completely correct answers in a significant number of cases. In particular, for these eight DRs, the answers for all six queries were correct, thereby each receiving a score of 2.0. The average score was less than 1.00 for only two DRs—DR17 and DR18. For the remaining eight DRs, the average score was at least 1.00, indicating that their analysis was correct, in part.

On the other hand, if we look across the queries, the average evaluation score for each query ranged between 1.16 and 2.00. In particular, DrAgent secured an average score of 1.50 for Q1. The fault type was correctly identified for all the DRs except three—DR10, DR17, and DR18. In other words, DrAgent (essentially, the LLM) was generally able to diagnose the fault type correctly in most cases. It may be recalled that we used Gemini without any domain-specific fine-tuning. An answer to Q1 was obtained by prompting the LLM with appropriate data and instructions. In other words, the results largely indicate that pre-trained LLMs, in general, can still offer useful results in specialized areas if prompted appropriately.

TABLE IV: Human evaluation scores for DrAgent

#	Q1	Q2	Q3	Q4	Q5	Q6	Average
DR1	2	2	2	2	2	2	2.00
DR2	2	2	2	2	2	2	2.00
DR3	2	2	2	2	2	2	2.00
DR4	2	2	2	2	2	2	2.00
DR5	2	2	0	2	2	2	1.66
DR6	2	2	0	2	2	2	1.66
DR7	2	2	2	2	2	2	2.00
DR8	2	2	2	1	2	2	1.83
DR9	2	2	1	2	2	2	1.83
DR10	-1	2	0	1	2	2	1.00
DR11	2	2	2	1	2	2	1.83
DR12	2	2	2	1	2	2	1.83
DR13	2	2	2	N/A	2	-1	1.40
DR14	2	2	2	N/A	2	-1	1.40
DR15	2	2	2	N/A	2	2	2.00
DR16	2	2	0	N/A	2	-1	1.00
DR17	-1	2	2	N/A	2	-1	0.80
DR18	-1	2	1	N/A	2	-1	0.60
Average	1.50	2.00	1.44	1.66	2.00	1.66	

On the other hand, the results for Q2 and Q5 were obtained using two different tools that performed computations with DR data. Since the underlying calculations are relatively complex for an LLM, tool usage resulted in the correct answers. However, comparable outcomes were not obtained for Q6, whose answers also were based on a user-defined tool. In general, the identification of fault clearance could be tricky sometimes. As such, our implementation of this function also failed to capture the diverse conditions. However, as noted in Table I, Q6 is of low importance, in general.

Table IV also shows that the average score for Q3 was 1.44. It may be noted that DrAgent answered Q3 by using a two-step process to identify the relay algorithms: (1) the names of the digital channels that were activated were identified by calling the get_digital_channels_activation_status tool and (2) the relay algorithms corresponding to the channel names were identified by looking at the product manuals using the search_product_manuals tool. In this case, correct answers were obtained for 12 out of 18 DRs. The results were partially correct for two DRs, DR9 and DR18. No useful answer was obtained for the remaining four DRs.

It may be noted that we used RAG to search the product manuals. Therefore, a likely reason for not receiving any meaningful answers for Q3 could be that the underlying retrieval mechanism had failed to retrieve sufficiently relevant context from the vector database. However, since vector search is often approximate, running the same query again could potentially lead to an answer. Nevertheless, in the future, it would be interesting to investigate alternative search and re-ranking mechanisms.

Figure 5 shows the evaluation scores, averaged across all the DRs, for the agentic approach using DrAgent and the non-agentic approach, as discussed in Section IV. The detailed scores for the non-agentic approach are shown in the Appendix.

The figure shows that DrAgent resulted in statistically significant better performance for Q2 and Q6, as compared

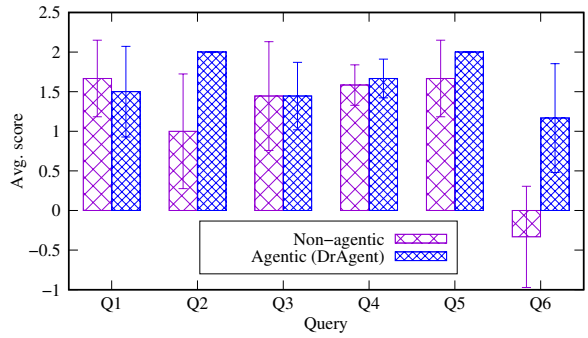

Fig. 5: Average performance of fault analysis using agentic and non-agentic approaches.

to the non-agentic approach. The result also appeared to be better in the case of Q5.

In the case of Q3, the average scores obtained using DrAgent and the non-agentic approach were the same, 1.44. However, as Figure 5 indicates, the confidence interval was narrower for DrAgent. In particular, the non-agentic approach resulted in four incorrect answers (see Appendix), while the remaining answers were correct. In contrast, DrAgent did not generate any wrong answers—they were either correct, partially correct, or no answer was generated—leading to a lower dispersion of the scores. It may be recalled that the non-agentic approach prompted the LLM with all the questions, data (text and images), and instructions in a single prompt. This potentially provided the LLM with a better context, which allowed it to interpret the digital channel names in a relatively better way. However, in general, no answer may be preferred over having incorrect answers.

Finally, in the case of Q1, the average score of DrAgent was marginally lower than the score obtained using a non-agentic approach. It may be noted that both approaches used the same data and instructions for fault-type diagnosis. Therefore, this variation in the results can be accounted for by the stochastic nature of LLMs.

We also evaluated DrAgent using other relevant queries and examined whether or not the correct functions were called. For example, Listing 1 shows the plan and tools called in response to the query "Show me the sequence of events as a table." Here, the get_digital_channels_activation_status function was called to get the activation status of the digital channels. Subsequently, the LLM was responsible for arranging the information in a tabular format.

On the other hand, the query "Analyze the fault in detail" led to the use of multiple tools, as shown in Listing 2. It may be noted that, unlike the other queries, this particular query was more generic. Accordingly, the LLM generated a plan to use some of the relevant tools available to it.

Listing 1: Plan of execution for the query "Show me the sequence of events as a table"

```
Get a summary of the digital channels and their activation
     status using the provided COMTRADE files.
#E1 = get_digital_channels_activation_status[cfg_file_path
     ='/app/dr.cfg', dat_file_path='/app/dr.dat']

Ask LLM to present the sequence of events from the digital
     channels activation status, as a table. Also, ask LLM
     to include the corresponding timestamps by considering
     the sampling rate of 1000 samples/sec.
#E2 = LLM[Task: Show me the sequence of events as a table.
     Context: Sampling rate = 1000 samples/sec. Digital
     channels activation status: #E1]
```

Listing 2: Plan of execution for the query "Analyze the fault in detail"

```
Find the fault type and involved phases using the
     find_fault_type function. This function will also
     provide an explanation of how the fault type was
     determined.
#E1 = find_fault_type[cfg_file_path='/app/dr.cfg',
     dat_file_path='/app/dr.dat']

Find the fault inception sample number using the
     find_fault_inception function.
#E2 = find_fault_inception[cfg_file_path='/app/dr.cfg',
     dat_file_path='/app/dr.dat',
     fault_inception_current_threshold=0.1,
     fault_inception_voltage_threshold=0.05]

Calculate the fault inception time in seconds.
#E3 = calculator[expression='#E2 / 1000.0']

Determine the maximum fault current using the
     find_max_fault_current function.
#E4 = find_max_fault_current[cfg_file_path='/app/dr.cfg',
     dat_file_path='/app/dr.dat']

Estimate the fault clearance sample number using the
     estimate_fault_clearance function.
#E5 = estimate_fault_clearance[cfg_file_path='/app/dr.cfg',
     dat_file_path='/app/dr.dat']

Analyze the fault based on the information gathered.
#E6 = LLM[Analyze the fault in detail, given the fault type:
      #E1, fault inception time: #E3 seconds, maximum fault
     current: #E4 Amperes, and fault clearance sample number
     : #E5. Consider providing recommendations for improving
      system protection based on the analysis.]
```

Figure 6 shows the tool usage time of two particular tools—find_fault_inception and estimate_fault_clearance—when the queries Q2, Q6, Q7, Q8, and Q9 were executed, in order, for DR2. Each of these two tools (function) took about half a second to execute[8] and generate the results in the concerned experimental environment. However, when Q8 and Q9 were executed, the outputs of these tools were retrieved from the cache, resulting in a tool usage time of about 0.1 milliseconds. In other words, the tool usage time in the latter case was reduced by around 5000 times. The results, therefore, illustrate that tool caching can be potentially useful, especially when fault analysis (or similar tasks) involves frequent usage of some tools and such functions are executed in an environment with low computational resources.

VI. Conclusion

Power grids are often subjected to faults. DR analysis is a critical and recurrent requirement to diagnose a fault and take appropriate action. However, fault analysis can be a challenging task for the non-experts. Moreover, when the scope

[8]Ideally, Q6 can be answered by using estimate_fault_clearance only. However, sometimes the LLM calculates the fault duration, thus executing find_fault_inception as well. In such scenarios, during our evaluation, we assigned the score of 2 if the numerical value of the duration was correct and the answer explicitly mentioned that it was fault duration.

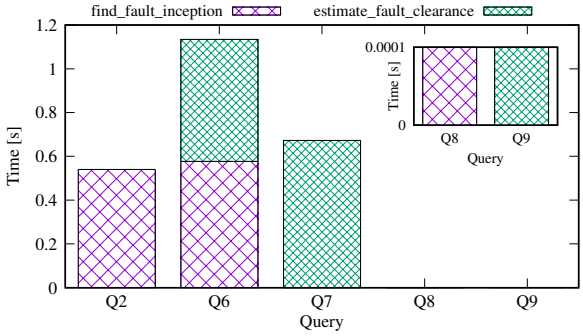

Fig. 6: Tool usage time for different queries for two particular tools, find_fault_inception and estimate_fault_clearance. Inset: a magnified view of the tool usage time for Q8 and Q9.

of analysis is expanded, it could become a time-consuming process. The emergence of GenAI and LLMs offers a new direction to address some of these challenges.

In this work, we discussed the design of DrAgent, which combines a pre-trained LLM's knowledge with domain-specific knowledge represented in the form of tools. In the agentic workflow, given a query, a plan is generated that uses one or more tools. Subsequently, the tools are executed to generate an answer. Such a step-by-step approach also enables a novice to learn new concepts and problem solving steps. It also allows the experts to verify the reasoning of the LLMs. The planning and execution are further improved using argument reconstruction and caching the tool outputs. Our experimental results indicated that DrAgent can, in general, offer better results—especially, when numerical accuracy is considered.

In the future, this work can be extended in different ways. Additional queries and tools may be considered to further expand the scope of fault analysis. On the other hand, the consideration of additional DRs may offer a more granular performance analysis. Finally, the scope of tool caching may be expanded to resource caching in order to reduce computational overhead and latency further.

References

[1] "IEEE Standard Common Format for Transient Data Exchange (COMTRADE) for Power Systems," *IEEE Std C37.111-1991*, pp. 1–28, 1991.

[2] A. Hoff, P. Lima, R. Bernardes, and R. Abboud, "Finding Faults Fast Saves Money and Improves Service," in *Western Protective Relay Conference*, Oct. 2021, pp. 1–14. [Online]. Available: https://wprcarchives.org/wp-content/uploads/2024/04/Taylor_Douglas_Whats-the-Risk-Avistas-Wildfire-Resiliency-Enhancements_2021.pdf

[3] O. D. Naidu and A. K. Pradhan, "Model Free Traveling Wave Based Fault Location Method for Series Compensated Transmission Line," *IEEE Access*, vol. 8, pp. 193 128–193 137, 2020.

[4] B. K. Saha, "Generative Artificial Intelligence for Industry: Opportunities, Challenges, and Impact," in *Proceedings of 2024 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2024, pp. 81–86.

[5] J. Zhang, J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang, B. Zheng, B. Liu, Y. Luo, and C. Wu, "Aflow: Automating agentic workflow generation," 2024. [Online]. Available: https://arxiv.org/abs/2410.10762

[6] B. Xu, Z. Peng, B. Lei, S. Mukherjee, Y. Liu, and D. Xu, "ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models," *arXiv preprint arXiv:2305.18323*, 2023.

[7] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in *International Conference on Learning Representations (ICLR)*, 2023.

[8] B. K. Saha and A. A. Parapurath, "Artificial Intelligence-based Segregation of Power Grid Disturbance Records," in *Proceedings of 2024 International Conference on Electrical, Computer and Energy Technologies (ICECET)*. IEEE, Jul. 2024.

[9] J. Hong, Y.-H. Kim, H. Nhung-Nguyen, J. Kwon, and H. Lee, "Deep-Learning Based Fault Events Analysis in Power Systems," *Energies*, vol. 15, no. 15, 2022. [Online]. Available: https://www.mdpi.com/1996-1073/15/15/5539

[10] Y. Chen, X. Fan, R. Huang, Q. Huang, A. Li, and K. P. Guddanti, "Artificial Intelligence/Machine Learning Technology in Power System Applications," United States, techreport, Apr. 2024. [Online]. Available: https://www.osti.gov/biblio/2340760

[11] R. Gitzel, M. W. Hoffmann, P. Z. Heiden, A. Skolik, S. Kaltenpoth, O. Müller, C. Kanak, K. Kandiah, M.-F. Stroh, W. Boos, M. Zajadatz, M. Suriyah, T. Leibfried, D. S. Singhal, M. Bürger, D. Hunting, A. Rehmer, and A. Boyaci, "Toward Cognitive Assistance and Prognosis Systems in Power Distribution Grids—Open Issues, Suitable Technologies, and Implementation Concepts," *IEEE Access*, vol. 12, pp. 107 927–107 943, 2024.

[12] T. Xiao and P. Xu, "Exploring automated energy optimization with unstructured building data: A multi-agent based framework leveraging large language models," *Energy and Buildings*, vol. 322, p. 114691, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378778824008077

[13] S. Kim, S. Moon, R. Tabrizi, N. Lee, M. W. Mahoney, K. Keutzer, and A. Gholami, "An LLM Compiler for Parallel Function Calling," in *International Conference on Machine Learning*, 2024.

[14] B. K. Saha, P. Gordon, and T. Gillbrand, "NLINQ: A natural language interface for querying network performance," *Applied Intelligence*, vol. 53, pp. 28 848–28 864, 2023.

[15] S. Baccianella, "JSON Repair - A python module to repair invalid JSON, commonly used to parse the output of LLMs," Oct. 2024. [Online]. Available: https://github.com/mangiucugna/json_repair

[16] G. Team *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," 2024. [Online]. Available: https://arxiv.org/abs/2403.05530

[17] P. Damodaran, "FlashRank, Lightest and Fastest 2nd Stage Reranker for search pipelines." Dec. 2023. [Online]. Available: https://github.com/PrithivirajDamodaran/FlashRank

Appendix
Human evaluation scores for the non-agentic approach of DR analysis

#	Q1	Q2	Q3	Q4	Q5	Q6	Average
DR1	2	-1	-1	2	2	-1	0.50
DR2	2	2	2	1	2	-1	1.33
DR3	2	2	-1	2	2	-1	1.00
DR4	2	2	2	1	2	-1	1.33
DR5	-1	2	2	1	2	-1	0.83
DR6	2	2	2	1	2	2	1.83
DR7	2	2	-1	2	2	-1	1.00
DR8	2	-1	-1	2	2	-1	0.50
DR9	2	2	2	2	2	-1	1.50
DR10	2	2	2	2	2	2	2.00
DR11	2	2	2	1	-1	2	1.33
DR12	2	2	3	2	2	2	2.16
DR13	2	2	2	N/A	2	-1	1.40
DR14	2	-1	2	N/A	2	-1	0.80
DR15	2	-1	2	N/A	-1	-1	0.20
DR16	2	2	3	N/A	2	-1	1.60
DR17	2	-1	2	N/A	2	-1	0.80
DR18	-1	-1	2	N/A	2	-1	0.20
Average	1.66	1.00	1.44	1.58	1.66	-0.33	