IOAgent: Democratizing Trustworthy HPC I/O Performance Diagnosis Capability via LLMs

Chris Egersdoerfer[1], Arnav Sareen[2], Jean Luca Bez[3], Suren Byna[4], Dongkuan (DK) Xu[5], and Dong Dai[1]

[1]University of Delaware, USA, {cegersdo, dai}@udel.edu
[2]University of North Carolina at Charlotte, USA, asareen2@charlotte.edu
[3]Lawrence Berkeley National Laboratory, USA, jlbez@lbl.gov
[4]The Ohio State University, USA, byna.1@osu.edu
[5]North Carolina State University, USA, dxu27@ncsu.edu

2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3315-3237-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/IPDPS64566.2025.00036

***Abstract*—As the complexity of the HPC storage stack rapidly grows, domain scientists face increasing challenges in effectively utilizing HPC storage systems to achieve their desired I/O performance. To identify and address I/O issues, scientists largely rely on I/O experts to analyze their I/O traces and provide insights into potential problems. However, with a limited number of I/O experts and the growing demand for data-intensive applications, inaccessibility has become a major bottleneck, hindering scientists from maximizing their productivity.**

The recent rapid progress in large language models (LLMs) opens the door to creating an automated tool that democratizes trustworthy I/O performance diagnosis capabilities to domain scientists. However, LLMs face significant challenges in this task, such as the inability to handle long context windows, a lack of accurate domain knowledge about HPC I/O, and the generation of hallucinations during complex interactions.

In this work, we propose `IOAgent` as a systematic effort to address these challenges. IOAgent integrates various new designs, including a module-based pre-processor, a RAG-based domain knowledge integrator, and a tree-based merger to accurately diagnose I/O issues from a given Darshan trace file. Similar to an I/O expert, IOAgent provides detailed justifications and references for its diagnoses and offers an interactive interface for scientists to continue asking questions about the diagnosis. To evaluate IOAgent, we collected a diverse set of labeled job traces and released the first open diagnosis test suite, `TraceBench`. Based on this test suite, extensive evaluations were conducted, demonstrating that IOAgent matches or outperforms state-of-the-art I/O diagnosis tools with accurate and useful diagnosis results. We also show that IOAgent is not tied to specific LLMs, performing similarly well with both proprietary and open-source LLMs. We believe IOAgent has the potential to become a powerful tool for scientists navigating complex HPC I/O subsystems in the future.

I. Introduction

High-performance computing (HPC) clusters play a crucial role in enabling modern science, supporting a wide range of scientific applications and services across various domains, such as cosmology [1], quantum simulation [19], astronomy [17], climate modeling [22], and life sciences [16].

In recent years, these scientific applications have become increasingly data-intensive, necessitating the efficient utilization of HPC I/O subsystems to meet performance requirements. However, effectively leveraging the complex I/O stacks in HPC systems, which include high-level parallel I/O libraries (e.g., HDF5 [13], PnetCDF [24]), API interfaces (e.g., MPI-IO, POSIX, STDIO), and file systems (e.g., Lustre [34], BurstBuffer [14]), remains a challenging task for most domain scientists. Large-scale scientific applications often experience slow I/O performance due to inefficient use of data access APIs, misconfigurations, or the failure to apply key optimizations available in the storage systems [36, 40].

One proven approach to assist scientists in optimizing their I/O performance is to record the I/O traces of their applications for post-hoc analysis [10, 52]. Real-time I/O profiling tools, such as Darshan [7] and Recorder [44], have been developed and deployed in modern HPC facilities to collect detailed I/O traces. These traces provide comprehensive information about the I/O behavior of applications, including access patterns, I/O sizes, and timing data to help scientists understand their application's I/O behaviors. Additional tools, such as PyDarshan [31] and DXT-Explorer [4], have been introduced to interpret these traces, identify potential I/O issues, and even offer optimization suggestions [11, 51].

Despite these advancements, diagnosing I/O issues from application traces often requires the involvement of human I/O experts, who possess the specialized knowledge needed to interpret trace data due to the inherent complexity of modern HPC storage systems. With the increasing number of scientists from various domains developing HPC applications, the shortage of readily available I/O expertise presents a significant barrier. Our interviews with I/O experts from NERSC also confirm that a substantial backlog of applications with I/O performance issues is frequently awaiting analysis, with no readily available means to absolve such complications.

This gap is further exacerbated by the growing complexity of HPC systems and the vast amounts of data generated by modern applications. As a result, the difficulty in identifying and resolving I/O performance bottlenecks limits the potential of scientists and leads to inefficient use of computational resources. Therefore, there is an urgent need for an automated tool that can democratize access to I/O optimization expertise.

Recent developments in large language models (LLMs) like ChatGPT and Claude have shown some promises in addressing such challenges. Pre-trained on large datasets, these models demonstrate ingenious abilities in understanding and generating human-like text, making them highly accessible. Their in-context learning ability [38], combined with

1530-2075/25/$31.00 ©2025 IEEE
DOI 10.1109/IPDPS64566.2025.00036

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

prompt engineering techniques [6] and Chain-of-Thought (CoT) reasoning [47], enables them to follow instructions and incorporate new information. Real-world examples span areas such as medical questioning [28], military simulation and strategy [37], log-based anomaly detection [11], and educational guidance [48]. Furthermore, recent advancements in Retrieval-Augmented Generation (RAG) allow LLMs to integrate external information from knowledge bases during the generation process [23], significantly enhancing their ability to produce accurate or contextually appropriate responses by incorporating information beyond their pre-training. The capability of continuous interactions between LLM and users is another unique capability, making it extremely useful as an assistant in many tasks.

These distinctive and compelling capabilities of LLMs illuminate a path to democratize domain scientists' access to HPC I/O optimization. Given an I/O trace, LLMs may automate the I/O performance diagnostic process, making HPC I/O optimization expertise more accessible to domain scientists and removing the necessity for a human expert. This can ultimately accelerate scientific discovery and enhance computational efficiency.

However, realizing such an ambitious goal is not straightforward, with traditional LLMs facing multiple significant challenges that inhibit their ability to interpret, analyze, and process I/O traces. First, although traces such as those generated by Darshan can be parsed into a human-readable format and hence are directly interpreted by LLMs, their lengths often surpass millions of lines which exceeds typical LLMs' context window size. Due to this, directly querying LLMs with such traces leads to a multitude of problems and in many cases, an inability to leverage the model entirely. For example, LLMs are known to encounter issues with long context windows, such as *lost-in-the-middle truncation*, where LLMs truncate the input, resulting in much of the information contained within center of the context being ignored in favor of text at extremities of the document [26], and *losing long-range dependencies*, where LLMs struggle to maintain coherence across distant parts of the input, preventing them from effectively modeling dependencies across lengthy documents [5]. Unfortunately, in practice, critical information about I/O issues could be located anywhere in the trace, such as inefficient I/O behavior for a subset of files in the middle of the application execution. Additionally, many I/O issues can only be identified by correlating multiple parts of the I/O trace, such as the amount of MPI-IO vs. the amount of POSIX IO, making it challenging, even for state-of-the-art LLMs to accurately diagnose I/O problems, as exemplified in the next section.

Second, diagnosing I/O performance issues from I/O traces requires extensive domain knowledge, as these issues may be deeply rooted in specific I/O libraries or embedded within the nuances of storage systems. However, such domain-specific knowledge might not be available to LLMs during their training stage, especially considering that new findings are continuously being published. Even if domain knowledge is available to LLMs during training, it may not be effectively utilized due to the vast corpus that these models are exposed to, which typically encompasses an extensive amount of general information, thus diluting the specificity necessary to effectively tackle domain-specific tasks (as demonstrated in an example later). While this generality is typically a prominent strength of modern LLMs, it significantly limits its capability to act with expert-level proficiency in a particular domain.

Third, LLMs are known to produce hallucinations, plausible but factually incorrect or non-sensical outputs, which are unacceptable for use as a diagnosis tool by domain scientists [18]. Although opting for more advanced models such as GPT-4o instead of simpler versions such as GPT-4 or open-source models like Llama [42], can help reduce hallucinations [55], larger models consequently introduce higher costs [33]. As a tool targeting general scientists working at the HPC scale, relying on expensive models is impractical. Therefore, addressing the hallucination challenge independently of an individual model is crucial to enable an LLM-based tool at the extensive magnitude commonly present in HPC applications.

In this study, we explore the feasibility and strategy of using large language models to provide trustworthy, expert-level HPC I/O performance diagnosis, thereby effectively guiding domain scientists in optimizing their I/O performance. To this end, we introduce `IOAgent`, an LLM-based agent designed to analyze applications' I/O traces. IOAgent incorporates key new designs to address the three challenges discussed earlier. First, IOAgent implements a module-based trace preprocessing strategy that groups relevant I/O activities together, preventing LLMs from truncating vital context or missing key I/O modules. Additionally, IOAgent introduces a set of summarization methods that accurately extract needed metadata from the trace file, rather than relying solely on the limited capabilities of LLMs for metadata retrieval. Second, IOAgent utilizes Retrieval-Augmented Generation (RAG) to integrate expert-level I/O knowledge into the diagnosis process. This allows IOAgent to provide references for its diagnoses, helping to avoid popular but incorrect claims while also increasing the transparency of the diagnosis process. Third, IOAgent employs a tree-based merging mechanism with self-reflection to minimize hallucinations in its diagnosis. These mechanisms enable IOAgent to provide domain scientists with accurate diagnostic summaries of I/O issues based on Darshan I/O traces, akin to consulting with a human expert, even if the scientists utilize open-source LLMs such as Llama.

To evaluate IOAgent and similar tools that may be developed in the future, we created a new benchmark suite for I/O issue diagnosis, named `TraceBench`. TraceBench includes over 40 Darshan traces collected from both I/O benchmarks and real-world applications, with each trace annotated with issues identified by I/O experts, and each I/O issue confirmed by at least two experts. By evaluating the diagnostic results with these standard labels, one can assess and compare the quality of diagnostic tools from a variety of sources. Using `Tracebench`, we demonstrated that IOAgent outperforms

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

state-of-the-art I/O diagnostic tools, such as Drishti [3] and ION [9] in terms of accuracy, clarity, and coverage.

The core value of IOAgent lies not only in diagnosing HPC I/O performance issues but also in its ability to provide valuable design choices on how to leverage powerful but difficult-to-manage LLMs for production-level systems, where both accuracy and cost are of utmost importance. We believe that similar tools can be developed to democratize other optimization capabilities in the HPC environment, such as computation and networking in the near future.

II. Background and Related Work

In this section, we provide an overview of HPC I/O profiling tools, focusing on Darshan and its extensions. We also discuss how researchers have utilized these tools to understand HPC I/O characteristics and its typical workflow. Subsequently, we introduce Drishti as a state-of-the-art I/O issue diagnosis tool. Finally, we provide background knowledge on large language models, their reasoning capabilities, and the concept of Retrieval-Augmented Generation (RAG).

A. Darshan: HPC I/O Profiling Tool

Efficient I/O performance is critical for HPC applications, and as a result, understanding I/O behavior is essential for optimization. Multiple I/O profiling tools have been developed to facilitate this understanding, including STAT [2], mpiP [43], IOPin [20], Recorder [44], and Darshan [7]. Among these, Darshan has gained widespread adoption in the HPC community due to its lightweight design and holistic descriptions of applications [32].

Darshan operates by instrumenting HPC applications to record key statistical metrics related to their I/O activities. In particular, it traces essential information for each file accessed across different I/O interfaces, including POSIX (Portable Operating System Interface) I/O, MPI (Message Passing Interface) I/O, and Standard I/O. The metrics collected encompass multiple aspects and can be mainly summarized as follows: `1) data volume`, amount of data read from and written to each file, 2) `operation counts`, number of read, write, and metadata operations performed by the application, 3) `temporal information`, aggregate time spent on read, write and metadata operations, 4) `rank information`, identification of the MPI ranks issuing I/O requests, and 5) `variability metrics`, variance of I/O sizes and timing among different application ranks.

Darshan also collects file system-specific metrics, such as Lustre stripe widths and Object Storage Target (OST) IDs over which a file is striped. This comprehensive data provides valuable insights into the I/O patterns and performance characteristics of HPC applications.

Darshan eXtended Tracing (DXT) [49] is a recent development based on Darshan to capture fine-grained records of an application's I/O operations, covering details such as specific files involved in I/O operations, each read/write operation, offset, and length of each I/O request, the start and end times of each I/O operation, and MPI rank IDs. Researchers have utilized Darshan DXT to analyze I/O behaviors across a variety of applications and systems [35]. However, since Darshan DXT introduces more noticeable overheads to HPC applications, it is typically not enabled by default. Hence, in this study, we focus only on the original Darshan I/O traces and leave working with Darshan DXT traces as future work.

B. Advanced I/O Issue Diagnosis: Drishti and ION

While tools like Darshan provide the raw data necessary for I/O analysis, interpreting this data to diagnose performance issues remains challenging. Several tools have been developed to address this gap, including IOMiner [45], UMAMI [30], TOKIO [29], DXT-Explorer [4], recorder-viz [44], and Drishti [3]. Of these, Drishti represents the most recent advancement in I/O issue diagnosis.

Drishti takes a Darshan trace as input and conducts an analysis to report various I/O performance issues based on a set of heuristic-based triggers. Currently, Drishti includes a set of 30 triggers corresponding to different application behaviors and can identify nine distinct types of I/O issues, such as `small I/O operations` (excessively small read/write requests), `misaligned I/O` (I/Os not aligned with the file system's block size), and `imbalanced I/O` (uneven distribution of I/O workload across ranks).

Due to its lightweight design of checking key triggers in Darshan traces, Drishti can be an effective tool for system administrators or scientists to quickly scan a large number of traces and identify applications with I/O issues. However, it has notable limitations when it comes to providing detailed I/O diagnoses for domain scientists working on their applications. First, Drishti relies on hard-coded threshold values (determined via expert knowledge and observations from past experience) for its triggers, which may not be accurate for all applications. For instance, it flags write requests smaller than 1MB as small writes and raises an issue if more than 10% of all requests are small I/Os. While 10% is a reasonable threshold for general applications, this can be perplexing to domain scientists if their application only exhibits a minute percentage of such operations, with the resulting performance degradation being negligible. Domain scientists would benefit far more from a tool that provides personalized, per-application explanations for each diagnosis. Second, Drishti's explanations and recommendations are predefined, hard-coded messages tied to their specific triggers. This approach lacks the nuanced and context-specific reasoning needed for domain scientists to fully understand and address performance problems. Lastly, Drishti does not offer an interactive interface for users to ask follow-up questions or further explore the analysis. This reduces its utility as a learning and exploratory tool, especially for domain scientists who may not have extensive I/O expertise.

These limitations motivated ION [9], a recent study that also uses large language models to diagnose HPC I/O issues. As a proof-of-concept work, ION directly queries the LLMs with engineered prompts to generate diagnoses. This strategy, however, means ION will heavily rely on the capability of the selected LLMs as well as suffer from their shortcomings, which will be further discussed later in Section III.

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

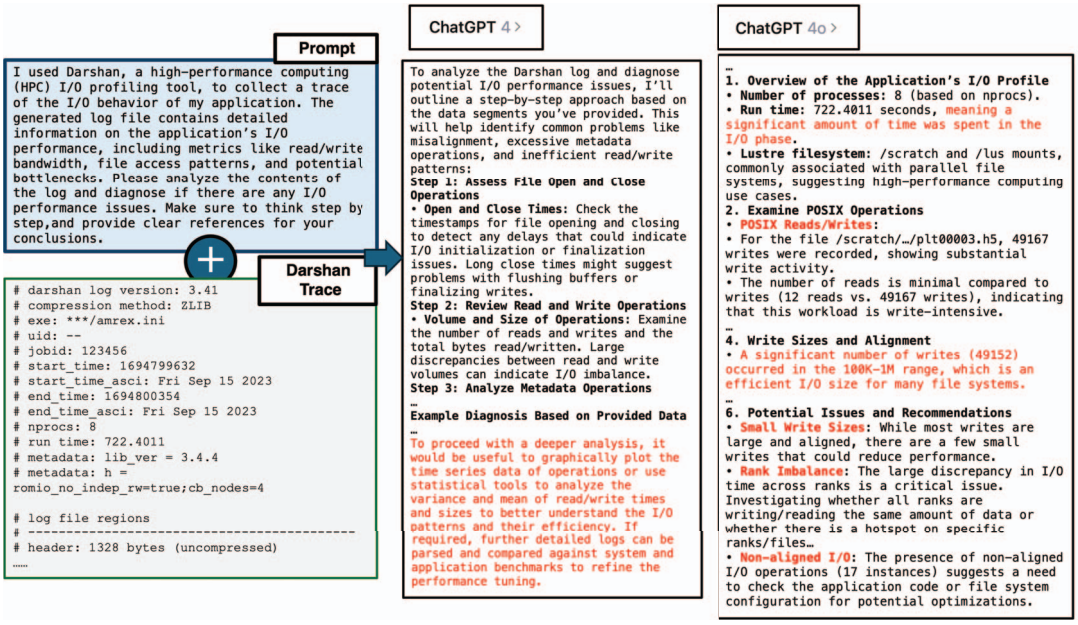


Fig. 1: An example diagnosis done by directly querying two LLMs: *gpt-4* and *gpt-4o*.

C. Large Language Models and Their Capabilities

Recent advancements in artificial intelligence have led to the development of large language models (LLMs) like ChatGPT, Claude AI, and Gemini [41], which demonstrate remarkable capabilities in understanding and generating humanesque text. One of the key strengths of LLMs is their ability to follow instructions and address various tasks using their *in-context learning* capability. They can perform complex tasks such as summarization [53], translation [6], and question answering [8], and even exhibit emergent abilities like mathematics and programming reasoning [46]. This makes them well-suited for applications that require comprehension and synthesis of information across diverse domains.

Another significant strength of LLMs is their enhancement via the utilization of *Retrieval-Augmented Generation (RAG)* [23]. RAG combines LLMs with external knowledge bases or documents, enabling the model to retrieve relevant information during the generation process. This approach enhances the model's ability to provide accurate and up-to-date information, particularly in specialized domains where the model's pre-training data may be insufficient and advancements are made rapidly.

In the context of HPC I/O analysis, LLMs offer a promising opportunity to automate the diagnostic process. However, applying LLMs to interpret I/O trace data and pinpoint I/O issues requires addressing several critical challenges, as discussed earlier. Recognizing both the upside potential and challenges of applying LLMs to HPC I/O analysis, our goal in this study is to develop a solution that harnesses the strengths of LLMs while mitigating their limitations.

III. Preliminary Study: Plain LLM Diagnosis

In this section, we first report the diagnostic results of a real-world HPC application's Darshan trace using plain language models. For this, we collected the Darshan trace of AMReX running at the NERSC HPC center. AMReX [54] is a widely used framework for highly concurrent, block-structured adaptive mesh refinement. This AMReX execution took around 722 seconds to finish, used 8 processes, and read/write 11 files to a Lustre file system mounted at `/scratch`.

Since the original Darshan trace is in binary format, we first used `darshan-parser` to convert it into a text-based, human-readable format. We then queried various language models using the prompt shown in Figure 1. Due to space constraints, we omit the outputs of open-source models such as Llama [42], as the quality of their results is significantly lower compared to OpenAI's GPT models.

The outputs from the ChatGPT-4 model are not particularly useful. It primarily generates a simple plan to assess different aspects of the I/O traces, such as open/close operations, read-/write operations, metadata operations, and stripe patterns, among others. In the end, it provides high-level recommendations on what domain scientists should do for deeper analysis, which are hardly useful; they are still left with the burden of learning how to '*graphically plot the time series data of operations or use statistical tools…*' to uncover potential I/O issues.

Compared to ChatGPT-4, the new 4o model makes significant progress in addressing the I/O diagnosis problem, as shown by the output on the right of Figure 1. It follows the prompt's instructions accurately, checking the I/O details and providing corresponding diagnoses. Most of the diagnoses are accompanied by concrete data about the I/O operations, such as '*The file alignment was set at 1MB (1048576 bytes), which matches the common Lustre stripe size. This is optimal for minimizing the number of I/O requests on Lustre,*'.

Although promising, ChatGPT-4o's outputs actually con-

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

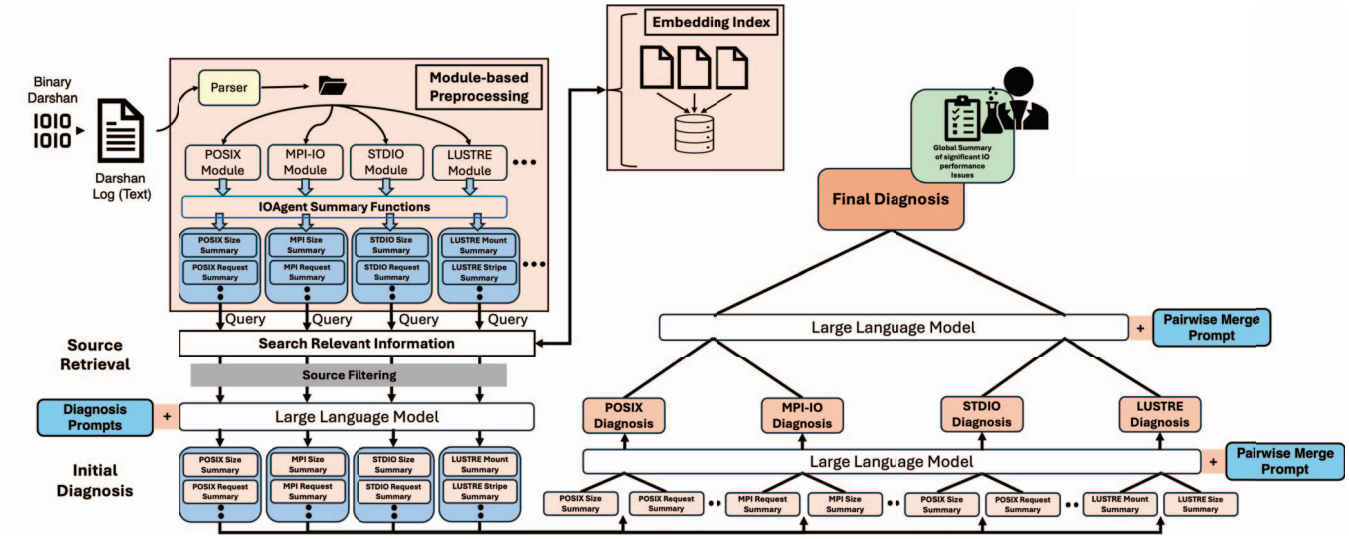


Fig. 2: The overall workflow of IOAgent and its key components.

tain multiple incorrect answers, an issue commonly seen with plain LLM-based agent tools [15]. First, due to context truncation, it misses or forgets important information in the I/O traces. In this AMReX run, a key I/O performance issue is the predominant use of the POSIX interface for I/O instead of MPI-IO, which is expected to offer better performance at this scale. The 4o model overlooks this issue, likely because the information about the MPI-IO model appears in the latter half of the Darshan trace. Second, the 4o model tends to overly rely on widely prevalent misconceptions that are believed but not necessarily true due to the broad-scoped nature of how LLMs are trained, which can lead to incorrect diagnoses. For example, it outputs "*...1MB...stripe size...is optimal for minimizing the number of I/O requests on Lustre*," but given that the `stripe count` is set to 1 in this application, such a configuration actually restricts parallel I/O capabilities, resulting in limited bandwidth and parallelism. Additionally, the "small write sizes" issue reported by the 4o model at the end may confuse domain scientists, as it previously reports "*a significant number of writes (49152) occurred in the 100K-1M range, which is an efficient I/O size*." This inconsistency in the diagnosis makes it difficult for domain scientists to fully trust or apply the tool.

We could not present the results from the latest OpenAI `o1-preview` model due to its limited context window, which is too small to process the complete AMReX Darshan trace. Our evaluations using smaller Darshan traces show that `o1-preview` produces outputs of similar quality to the 4o model. However, the high cost of `o1-preview` makes it largely impractical for our large-scale use.

These issues highlight the limitations of naively applying large language models to the I/O performance diagnosis task. We propose IOAgent, an LLM-based I/O diagnosis tool designed with key features to overcome these deficiencies.

IV. IOAgent Design and Implementation

In order to address the aforementioned challenges which non-augmented LLMs face when analyzing I/O trace logs, IOAgent mainly consists of three primary components. The first component, the *module-based pre-processor* primarily handles the context-length challenge faced by LLMs and guides IOAgent towards effective downstream source retrieval. The second component, the *Domain Knowledge Integrator* alleviates the non-augmented LLM's gaps in specific domain expertise with up-to-date and relevant information. The final component, the *Tree-Merge* (with self-reflection), enables complex summarization for both frontier and short-of-frontier language models to form comprehensive and interpretable I/O performance diagnoses.

A. Module-based Pre-processor

To address the challenge of integrating HPC trace logs into the limited context window of LLMs, IOAgent uses a log pre-processor based on two key insights. First, effective and comprehensive reasoning over a trace log requires access to data from all key modules used by the application. To meet this need, IOAgent's pre-processor separates the Darshan log into a set of CSV files, with each file containing the counters and values from a single Darshan module.

Second, since module data may also be extensive, it must be reduced to a manageable length for the LLM to interpret. IOAgent accomplishes this through summary extraction functions for each module, which generate categorized summary fragments. The categories of these summaries are outlined by the column titles in Table I. Due to the variance in counters accumulated by Darshan for each supported module, each module extracts varying categories of summary information. Due to these deviations, each module's summary category uses its own extraction function based on the available counters, but the information follows consistent principles across categories. For example, the I/O volume function for the STDIO module extracts the total amount of read and written bytes, while for LUSTRE, it focuses on information about mount points, stripe settings, and server usage.

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

Module	Summary Category									
	I/O Size	I/O Request Count	File	Metadata	Rank	Alignment	Order	Mount	Stripe Setting	Server Usage
POSIX	✓	✓	✓	✓	✓	✓	✓	-	-	-
MPIIO	✓	✓	✓	✓	✓	-	-	-	-	-
STDIO	✓	✓	✓	-	-	-	-	-	-	-
LUSTRE	-	-	-	-	-	-	-	✓	✓	✓

TABLE I: Coverage of Summary Categories Across Darshan Modules

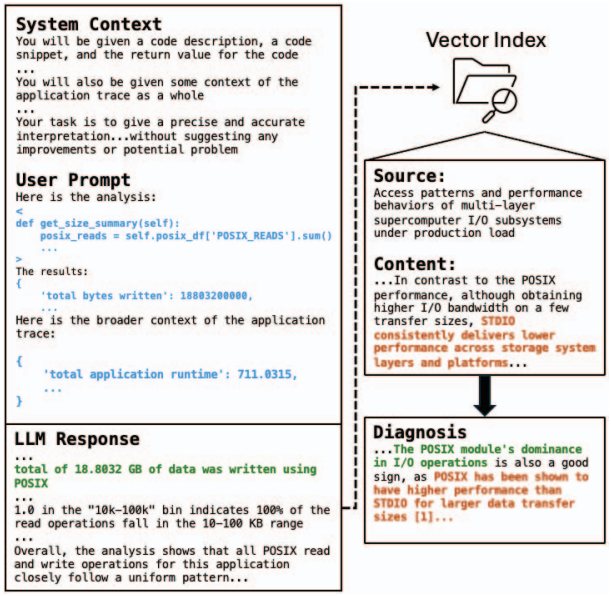


Fig. 3: An example prompt and response transforming JSON summary fragment to natural language

B. Domain Knowledge Integrator

IOAgent uses Retrieval-Augmented Generation (RAG) to integrate domain knowledge to provide trustworthy diagnosis for HPC I/O issues. There are three key questions while using RAG: 1) how to construct the queries, 2) how to build the vector database, 3) and what are the key vector search parameters. We describe each of them in detail below.

1) RAG Query Construction: RAG provides a database for the LLM to query and retrieve useful and relevant information for their current task. Hence, the quality of the query to the RAG database matters significantly to retrieve the needed information.

After module-based pre-processing, we split module information into CSV files and compress it into manageable JSON summary fragments, which helps avoid overwhelming the limited context of LLMs. Each JSON fragment represents specific information from a Darshan module, making it easier to locate relevant information using techniques like cosine similarity between text embeddings. However, JSON summaries are still not in the same format as expert knowledge, which is often communicated through research papers. This makes direct embedding searches over domain-specific knowledge ineffective.

To address this, IOAgent transforms each JSON summary into a natural language format for better alignment for the domain knowledge retrieval using language-based embedding similarity. This transformation is done by prompting an LLM with the code used to generate the summary, the JSON summary values, and a broader application context (total runtime, number of processes, and I/O size proportions from Darshan modules). An example of this transformation is shown in Figure 3, where the LLM provides a descriptive interpretation of the I/O size summary from the POSIX module. The natural language response aligns better with domain knowledge, improving the accuracy of embedding similarity searches. For instance, the JSON summary may contain: "*read_histogram: {'0-100': 1.0}*", but the corresponding natural language would become: "*...the value of 1.0 in the 1 to 100 bin indicates that 100% of the read operation fall within the 0 KB to 100 KB range*". The latter representation makes it much easier to match relevant studies in publications.

2) Vector Database Creation: The quality of the RAG database profoundly impacts the accuracy of subsequent diagnoses. However, to the best of our knowledge, there is no existing collection of text embeddings specifically targeting up-to-date HPC I/O performance diagnosis. Creating such a resource poses challenges, including the limits on how much data can be feasibly collected, embedded, and stored, which requires filtering and uncertainty about the best data sources. To address these issues and gather relevant, current knowledge on HPC I/O performance, we surveyed the past five years of research using the query 'HPC I/O Performance' and 'I/O Performance issue' in the ACM Digital Library and IEEE Xplore databases. From the top 50 results in each, we manually filtered for relevance, yielding 66 key works.

To process these works into a vector index for retrieval by IOAgent, we use LlamaIndex [25], a popular open-source framework for vector index creation. LlamaIndex allows for configuring common hyperparameters such as the embedding model, chunk size, and overlap between chunks. We found retrieval accuracy and relevance to be consistent with reasonable variations in these settings, so we used the default configuration: a chunk size of 512, an overlap of 20, measured distance between embeddings via cosine similarity, and the *text-embedding-3-large* model from OpenAI.

3) Vector Database Search: With the vector index implemented as described, IOAgent conducts a vector search to match specific, related domain knowledge to each summary fragment as shown in Figure 2. Since in many cases the most useful information available in the vector index for a given summary fragment may not be the most pertinent and potential additional context provided by other highly ranked but not highest ranked matches may still be incredibly useful for an accurate diagnosis, IOAgent retrieves the top 15 closest matches from the index. However, since this

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

Label	Description
High Metadata Load	The application spends a significant amount of time performing metadata operations (e.g., directory lookups, file system operations).
Misaligned [Read|Write] Requests	The application makes read or write requests that are not aligned with the file system's stripe boundaries.
Random Access Patterns on [Read|Write]	The application issues read or write requests in a random access pattern.
Shared File Access	The application has multiple processes or ranks accessing the same file.
Small [Read|Write] I/O Requests	The application is making frequent read or write requests with a small number of bytes.
Repetitive Data Access on Read	The application is making read requests to the same data repeatedly.
Server Load Imbalance	The application issues a disproportionate amount of I/O traffic to some servers compared to others or does not properly utilize the available storage resources.
Rank Load Imbalance	The application has MPI ranks issuing a disproportionate amount of I/O traffic compared to others.
Multi-Process Without MPI	The application has multiple processes but does not leverage MPI.
No Collective I/O on [Read|Write]	The application does not perform collective I/O on read or write operations.
Low-Level Library on [Read|Write]	The application relies on a low-level library like STDIO for a significant amount of read or write operations outside of loading/reading configuration or output files.

TABLE II: Table of I/O Issues and Descriptions

enlarges the context significantly for any subsequent query that should analyze the content of these retrieved sources together, we implemented a *self-reflection* step which uses a faster and cheaper language model (e.g., gpt-4o-mini) with less reasoning capability to rule out any sources which are found not to be relevant to the given summary fragment used as the original query over the index. This source filtering is run in parallel over all retrieved sources and rules out nearly half of the retrieved sources based on a more nuanced understanding of relevance than what is provided by the vector retriever. Following this parallel filtering step, IOAgent conducts its first true diagnosis of any potential I/O performance issues based on the information in the descriptive summary fragment generated by the LLM, as exemplified in Figure 3, and the filtered domain knowledge retrieved from the vector index.

C. Tree-based Merge

Once IOAgent has generated an initial diagnosis for each summary category in the Darshan modules using the retrieved domain knowledge, these diagnoses—and their relevant references—must be merged into a single, comprehensive I/O performance diagnosis for the entire application.

Intuitively, one could merge all of the fragmented diagnoses and their source content via a single LLM query, as it is possible that all of this information could fit within the context window of some larger models. However, as later evaluations will show, the task of effectively and consistently merging more than two diagnosis summaries is beyond the capabilities of existing LLMs, even proprietary ones. This is primarily because regardless of the size of an LLM's context window, the merging task itself requires diligent and precise reasoning. Effectively merging two diagnoses with their respective references involves removing redundant information, resolving contradictory details, and reflecting on both sets of provided sources to decide which should be retained in the new summary and where they should be cited. Additionally, adding any more summaries to the merging task beyond the minimal set of two introduces a more significant positional bias due to the increased cognitive load required in such a setting [26, 50].

In light of the challenges associated with merging diagnosis summaries, IOAgent implements a pairwise merging strategy, in which two diagnosis fragments and their domain knowledge references are merged into a new, combined diagnosis and set of references. Since all pairs of diagnoses merged at the same level of the tree are independent of each other, all merging tasks for each level are conducted in parallel. The partial summaries are then further merged, effectively forming a tree structure, as shown in Figure 2. This tree-based merging plays a critical role in delivering concise and accurate diagnoses, as later evaluation results demonstrate.

V. Benchmark Suite: TraceBench

Accurate evaluation of HPC I/O trace analysis tools presents a challenge due to the lack of publicly available trace datasets with labeled I/O performance issues. To address this, in line with our key contributions, we constructed the *TraceBench* dataset, which includes a set of Darshan traces from three different sources. Each trace has been manually labeled based on a predefined set of labels covering common HPC I/O performance issues, as represented in Table II.

The first data source consists of a set of Darshan logs generated by simple C source codes that intentionally introduce at least one of the I/O performance issues defined in the label set. We refer to this set as *Simple-Bench*. The second data source consists of a set of Darshan logs generated by the IO500 benchmark [21], where each configuration is designed to induce one or more I/O performance issues defined by the labels. The final data source primarily comprises real application traces, all collected on production HPC systems.

1) Simple-Bench (SB). The Simple-Bench set consists of 10 labeled Darshan logs, which were generated using 10 rudimentary scripts written in C to target at least one specific I/O performance issue category from Table II. However, as shown by the sample representation across different issue categories for each dataset in Table III, some traces from the Simple-Bench source contain more than one issue type. Despite this, the simplicity of these scripts results in Darshan trace logs that are small in size, with very low aggregate I/O volume and highly uniform behavior. As a result, these traces should be the easiest to diagnose accurately among all tools.

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

2) IO500. The second set of labeled Darshan trace logs was collected using the IO500 benchmark, a synthetic performance benchmark tool for HPC systems that simulates a broad set of commonly observed I/O patterns in HPC applications [21]. IO500 consists of many configurable workloads, which can be tuned to simulate various sub-optimal I/O patterns. For example, IO500's ior-easy workload, which conducts intense sequential read and write phases, can be tuned to use 8k transfer sizes issued through independent POSIX operations across multiple ranks, resulting in dominant small I/O behavior that does not effectively leverage higher-level libraries such as MPI-IO for multi-rank I/O. In total, 21 traces were collected from 21 unique configurations of IO500, and as shown in Table III, a significant number of traces exhibit multiple overlapping issues.

3) Real Applications (RA). The final set of labeled Darshan trace logs was generated by running real application workloads on large-scale production HPC systems. This collection consists of nine samples, each originating from a unique application, except for two samples representing recollected traces for the E2E and OpenPMD applications. In these cases, the original trace for each application possessed a significant performance issue, and the recollected traces had their primary issues resolved.

Labeled Issue	SB	IO500	RA	Total
High Metadata Load	1	2	2	5
Misaligned Read requests	2	10	4	16
Misaligned Write requests	2	10	6	18
Random Access Patterns on Write	0	5	2	7
Random Access Patterns on Read	0	5	2	7
Shared File Access	1	14	4	19
Small Read I/O Requests	2	10	5	17
Small Write I/O Requests	2	10	6	18
Repetitive Data Access on Read	1	0	0	1
Server Load Imbalance	7	15	2	24
Rank Load Imbalance	1	0	1	2
Multi-Process W/O MPI	0	13	0	13
No Collective I/O on Read	6	8	4	18
No Collective I/O on Write	5	8	2	15
Low-Level Library on Read	1	0	0	1
Low-Level Library on Write	1	0	0	1

TABLE III: Summary of traces and labeled issues.

Table III summarizes all the I/O issues included in `TraceBench` and how different sources, such as SB (Single-Bench), IO500, RA (Real-Application), contribute to these issues. There are 182 issues reported in 40+ traces.

VI. Evaluation

Summary. We conducted comprehensive evaluations of IOAgent using the `TraceBench` test suite. The diagnosis results were compared to those from the LLM-based diagnosis tool ION [9] and the expert-guided I/O performance diagnosis tool Drishti [3]. Additionally, we evaluated examples of advanced user interactions enabled by IOAgent's LLM-centric design and validated our tree-based merge design, as outlined in Section IV.

A. Evaluation Metrics

To evaluate IOAgent, an *accuracy* metric, which measures how well the diagnosis matches the ground truth, is certainly the most important criterion to consider. With this objective in mind, we expect IOAgent to perform on par with state-of-the-art expert-based diagnosis tools, such as Drishti. However, accuracy should not be the only critical metric evaluated. Since IOAgent is intended to assist domain scientists, the readability and understandability of the information provided in the diagnosis also become essential for users at any level of familiarity with HPC I/O. As an assistant, IOAgent should also be assessed based on how useful the information is. Following practices from the NLP community for evaluating agents [12, 27, 56], we propose the following three metrics for our evaluations:

- *Accuracy*: evaluate how accurately the ground truth labels are diagnosed by each tool.
- *Utility*: evaluate how useful the information provided in each diagnosis is for understanding the overall I/O behavior of the application, identifying performance issues, and determining how to address each noted issue (regardless of the factuality of such statements).
- *Interpretability*: evaluate how readable and understandable the provided information is for users at any level of familiarity with HPC I/O.

B. LLM-based Rating System

With the metrics defined, the challenge shifts to providing a quantitative judgment of each metric. Among them, *accuracy* is relatively easy as we can easily count the number of matched and mismatched issues. But *Utility* and *Interpretability* are rather subjective to individuals and their experience level with HPC I/O. In this study, we use LLM as the judge and follow the common practice of using a ranking-based system to compare different solutions.

The intuition behind using a capable LLM to rate the diagnostic results is simple. Earlier results in Figure 1 have shown a qualified LLM, such as GPT-4o, is very capable of understanding high-level I/O relevant concepts (with misunderstanding in many details of course), which highly emulates our target users: domain scientists. We believe that letting capable LLM serve as the judge avoids personal bias, and more importantly, brings in the perspectives of domain users.

Even with a capable LLM, quantitatively providing a score on *Utility* or *Interpretability* is still not feasible. Instead, we use an anonymized rating system to compare different solutions and rank them. The ranking is done by the LLM itself by formatting a prompt with all diagnosis outputs from different tools, a description of the specified evaluation criteria, and a description of how the ranking output should be formatted. The LLM, GPT-4o in this case, will rank the diagnosis outputs of different tools on a scale of 1 to 4, with 1 being the best and 4 being the worst.

Due to the known potential for *positional bias* in LLM-based content ranking [26, 39] we further augment the prompt in three key ways to ensure a fair and reliable ranking result. The first augmentation is to anonymize the

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

TABLE IV: Performance Results for Diagnosis Tools on TraceBench Subsets

Metric	Diagnosis Tool	Simple-Bench	IO500	Real-Applications	Overall
Accuracy	Drishti	0.398	0.480	0.472	0.459
	ION	0.343	0.381	0.417	0.380
	IOAgent-gpt-4o	**0.630**	**0.655**	**0.620**	**0.641**
	IOAgent-llama-3.1-70B	0.620	0.488	0.463	0.513
Utility	Drishti	0.426	0.417	0.491	0.436
	ION	0.352	0.401	0.380	0.385
	IOAgent-gpt-4o	0.565	**0.615**	**0.639**	**0.609**
	IOAgent-llama-3.1-70B	**0.694**	0.587	0.565	0.607
Interpretability	Drishti	0.417	0.452	0.463	0.447
	ION	0.343	0.417	0.352	0.385
	IOAgent-gpt-4o	0.546	**0.659**	**0.713**	**0.645**
	IOAgent-llama-3.1-70B	**0.694**	0.484	0.472	0.530
Average	Drishti	0.414	0.450	0.475	0.447
	ION	0.346	0.399	0.383	0.383
	IOAgent-gpt-4o	0.580	**0.643**	**0.657**	**0.632**
	IOAgent-llama-3.1-70B	**0.670**	0.520	0.500	0.550

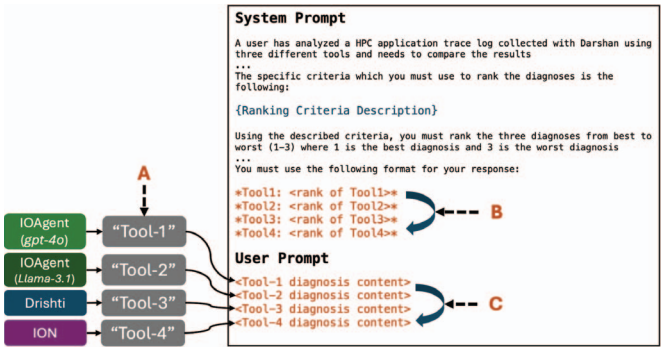


Fig. 4: A, B and C denote three augmentations applied.

names of the diagnosis tools to eliminate any potential bias towards or against specific analysis tools. The second augmentation is to rotate the rank assignment order imposed by the response formatting portion of the prompt. The final augmentation is to rotate the order in which the content of each diagnosis appears in the prompt. All these measures are intended to eliminate positional bias for each diagnosis in the prompt context. These augmentations are outlined in Figure 4, denoted by the letters A, B, and C. Additionally, for each sample, we rank the diagnoses four times, ensuring that every ranking prompt permutation appears at least once, further reducing potential bias.

C. Quantitative Score Calculation

Once the diagnoses have been ranked, we calculate the aggregated score using the following approach: For each trace log L, the diagnosis result from each diagnosis tool T will be evaluated three times on diagnosis criterion $C \in \{\text{Accuracy}, \text{Utility}, \text{Interpretability}\}$ and assigned a $Rank \in [1, 4]$. The score for each sample is calculated as $S_{T,C,L} = (4 - \text{Rank}_{T,C,L})$. A lower numerical rank (e.g. Rank 1) indicates a higher or better score.

The total score over all samples for a given data source $D \in \{\text{Simple-Bench}, \text{IO500}, \text{Real-Applications}\}$ is therefore defined as the sum of $S_{T,C,L}$:

$$S_{T,C,D} = \sum_{L \in D} S_{T,C,L} \tag{1}$$

We then normalize the score into a value between 0 and 1 by dividing it by the maximal score one can get:

$$NS_{T,C,D} = \frac{S_{T,C,D}}{(4-1) \cdot |D|} \tag{2}$$

Here $(4-1)$ is the highest score for each trace in D. Based on $NS_{T,C,D}$, we can define the average score of each tool across all three metrics over all data sources to show how the tool works across metrics. It is simply an average of $NS_{T,C,D}$ over all three metrics. Similarly, we also define the average score of each tool across all data sources on each metric to show how the tool works across different data sources.

Note that, to minimize hallucination by the LLM during ranking, we require the model to verify the reasoning behind the assigned positions by including a rank assignment explanation as part of the prompt. This approach provides significantly more insight into the assigned scores, as demonstrated by the following example of an explanation provided by the ranking LLM:

> Tool-2 and Tool-3 both provided comprehensive diagnoses that accurately identified all five I/O performance issues: small read and write I/O requests, misaligned read and write requests, and the use of multiple processes without MPI. Tool-2 provided detailed recommendations and emphasized the need to align with the file system boundaries, making it a strong contender for the top rank.

D. Ranking Results

Table IV summarizes all the results. We present the results for each metric (i.e., accuracy, utility, and interpretability) as well as the average across all metrics in the rows. The results for each type of job trace in `TraceBench`, along with the overall average, are shown in the columns. Each individual row represents one diagnosis tool, including the naive LLM-based tool (ION) using gpt-4o (*gpt-4o-2024-05-13*) as its

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

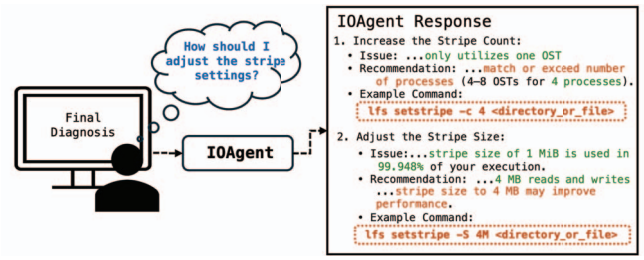


Fig. 5: Post-diagnosis example between user and IOAgent.

backbone and the state-of-the-art heuristic tool (Drishti). For IOAgent, we include two instances, labeled "IOAgent-*": one uses the proprietary gpt-4o (*gpt-4o-2024-05-13*) model from OpenAI, and the other uses the open-source *Llama-3.1-70B-Instruct* model from Meta. By including results from both instances, we want to evaluate that if IOAgent relies on specific proprietary models.

The results in Table IV clearly show that both IOAgent tools perform exceptionally well across all metrics and job traces. As expected, IOAgent-gpt-4o performs particularly well, given that it uses a frontier-level LLM; however, it is noteworthy that IOAgent-llama-3.1-70B also performs strongly. Its results are close to those of the gpt-4o model and surpass those of the state-of-the-art heuristic tool (Drishti) and the naive LLM-based tool (ION). Interestingly, Llama IOAgent seems to excel in more primitive cases like Simple-Bench. This may be due to IOAgent-gpt-4o providing too many details in such basic cases, making its output less useful or understandable from the user's perspective. These results confirm that our carefully designed IOAgent workflow makes LLMs a stable and practical tool for assisting scientists in understanding their applications' I/O issues.

E. Continued User Interaction

A primary feature enabled by the LLM-centric design of IOAgent is the ability for users to gain a deeper understanding and receive application-specific recommendations with implementation guidance through continued chat-based interaction. To evaluate this feature, we present a real interaction case summarized in Figure 5. In this example, IOAgent analyzed a log from the IO500 `TraceBench` subset, which performed a significant number of 4MB reads and writes but used the default Lustre stripe width and stripe size parameters of 1 and 1MB, respectively. The final diagnosis highlighted these potentially suboptimal stripe settings.

Following the diagnosis, the user simply prompted IOAgent about how to fix such an issue (highlighted in blue). In this case, IOAgent effectively utilized the context of the diagnosis and its referenced sources to provide detailed assistance, offering explanations of recommended actions and code samples, such as `lfs setstripe -S 4M` (highlighted in orange). These responses were not only accurate but also tailored to the specific issue identified by the application (highlighted in green). Such cases demonstrate IOAgent's unique potential to effectively assist domain scientists at scale.

F. Why Tree-based Merge?

The *tree-based merge* mechanism introduces both additional time and monetary overhead, as merging is done in pairs rather than all at once through a single prompt. In this evaluation, we aim to demonstrate the necessity of this overhead. Specifically, we present a comparison of using the tree-based merge versus not using it, as shown in Figure 6. For this benchmark, we used a less capable open-source *Llama-3-70B* model to illustrate how direct merging can be problematic. Due to space constraints, we use the example of merging just four summary diagnoses: *Size*, *Request Count*, *Metadata*, and *Request Order*. At the bottom of Fig. 6, we briefly list the issues identified with each category. We then compare the merged results of our tree-based approach with the naive one-step solution, using the same prompts.

The results of the 1-step merge show that important information regarding the non-sequential I/O patterns and insight into the stride sizes as well as the specific recommended use of higher level parallel I/O libraries (MPI-IO) are all lost, along with their respective domain reference sources. In contrast, the tree-based merge successfully maintains the key points of each individual summary diagnosis as well as the useful domain reference sources pertaining to each. Note that, this is just a simpler case where only four summaries need to be merged. In IOAgent, we typically need to deal with 13 summary diagnoses, which is extremely challenging even for the latest gpt-4o model based on our experiments.

VII. Conclusion and Future Work

In this study, we propose and implement IOAgent, a model-agnostic LLM-based framework for HPC I/O trace log analysis which fills the existing gaps faced by non-augmented language models while analyzing logs. IOAgent implements a module-based log pre-processor, a vector index for domain information retrieval, and a tree-based summarization strategy to enable accurate, context-aware, and user-friendly I/O performance analysis achieving results on-par with or superior to existing, expert-level tools as evaluated on multiple criteria. Leveraging the conversational strengths of language models further democratizes access to interactive and accurate I/O performance diagnoses. Our future work includes the expansion of IOAgent's existing capabilities to provide in-depth user interaction capabilities, such as building the continually advancing reasoning capabilities of LLMs and the integration of continual analysis improvements through interaction.

Acknowledgments

We sincerely thank the anonymous reviewers for their valuable feedback. This work was supported in part by National Science Foundation (NSF) under grants CNS-2008265, CCF-2412345, OAC-2417850, and BCS-2416846. This effort was also supported in part by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research (ASCR) under contract number DE-AC02-05CH11231 with LBNL and under an LBNL subcontract to OSU (GR130493).

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

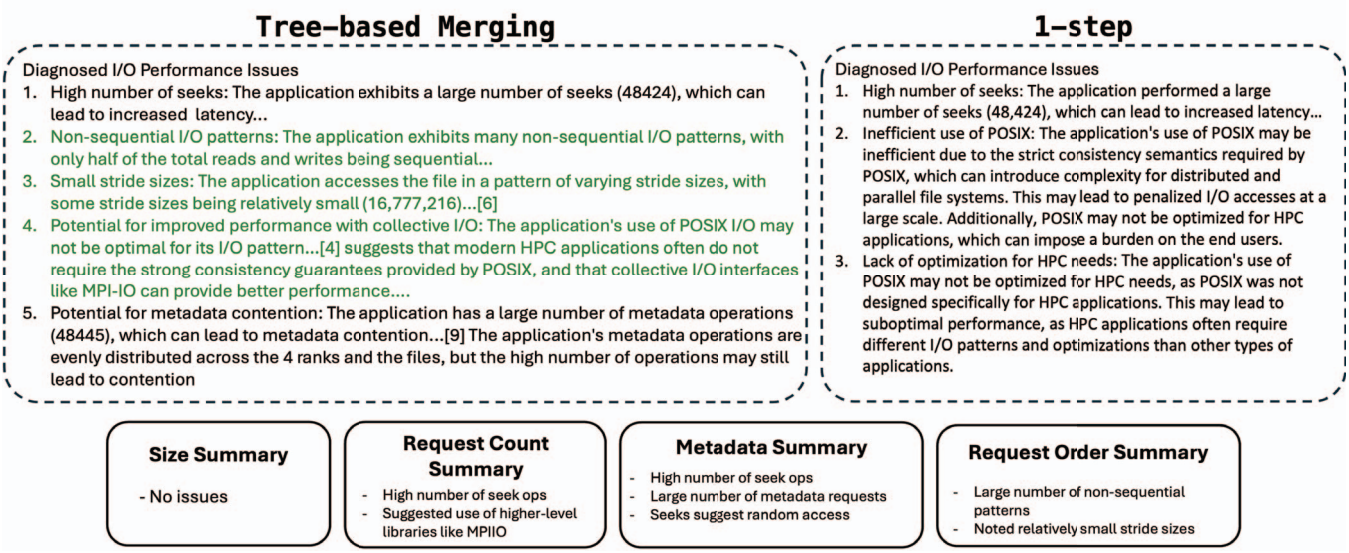


Fig. 6: Comparison of output using pair-wise vs 1-step merge between 4 diagnosis summaries

REFERENCES

[1] Ann S Almgren, John B Bell, Mike J Lijewski, Zarija Lukić, and Ethan Van Andel. Nyx: A massively parallel amr code for computational cosmology. *The Astrophysical Journal*, 765(1):39, 2013.

[2] Dorian C. Arnold, Dong H. Ahn, Bronis R. de Supinski, Gregory L. Lee, Barton P. Miller, and Martin Schulz. Stack trace analysis for large scale debugging. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, 2007.

[3] Jean Luca Bez, Hammad Ather, and Suren Byna. Drishti: Guiding end-users in the i/o optimization journey. In *2022 IEEE/ACM International Parallel Data Systems Workshop (PDSW)*, pages 1–6, 2022.

[4] Jean Luca Bez, Houjun Tang, Bing Xie, David Williams-Young, Rob Latham, Rob Ross, Sarp Oral, and Suren Byna. I/O Bottleneck Detection and Tuning: Connecting the Dots using Interactive Log Analysis. In *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)*, pages 15–22, 2021.

[5] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens, 2022.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[7] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Trans. Storage*, 7(3), oct 2011.

[8] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. A survey on evaluation of large language models, 2023.

[9] Chris Egersdoerfer, Arnav Sareen, Jean Luca Bez, Suren Byna, and Dong Dai. Ion: Navigating the hpc i/o optimization journey using large language models. In *Proceedings of the 16th ACM Workshop on Hot Topics in Storage and File Systems*, pages 86–92, 2024.

[10] Chris Egersdoerfer, Di Zhang, and Dong Dai. Clusterlog: Clustering logs for effective log-based anomaly detection. In *2022 IEEE/ACM 12th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*, pages 1–10. IEEE, 2022.

[11] Chris Egersdoerfer, Di Zhang, and Dong Dai. Early exploration of using chatgpt for log-based anomaly detection on parallel file systems logs. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, pages 315–316, 2023.

[12] ExplodingGradients. *Ragas Documentation*, 2023. Accessed: 2024-10-10.

[13] The HDF Group. The hdf5® library & file format, 1997-.

[14] Dave Henseler, Benjamin Landsteiner, Doug Petesch, Cornell Wright, and Nicholas J Wright. Architecture

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

and design of cray datawarp. *Cray User Group CUG*, 2016.

[15] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, 2023.

[16] Pegasus ISI. 1000genomes workflow, 2023. Accessed: 2024-08-04.

[17] Joseph C Jacob, Daniel S Katz, G Bruce Berriman, John C Good, Anastasia Laity, Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su, Thomas Prince, et al. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering*, 4(2):73–87, 2009.

[18] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, March 2023.

[19] Jeongnim Kim, Andrew D Baczewski, Todd D Beaudet, Anouar Benali, M Chandler Bennett, Mark A Berrill, Nick S Blunt, Edgar Josué Landinez Borda, Michele Casula, David M Ceperley, et al. Qmcpack: an open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules and solids. *Journal of Physics: Condensed Matter*, 30(19):195901, 2018.

[20] Seong Jo Kim, Seung Woo Son, Wei-keng Liao, Mahmut Kandemir, Rajeev Thakur, and Alok Choudhary. Iopin: Runtime profiling of parallel i/o in hpc systems. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 18–23, 2012.

[21] Julian M. Kunkel, John Bent, Jay Lofstead, and George S. Markomanolis. Establishing the io-500 benchmark. White Paper, the IO500 Foundation, Tech, 2016. [Online]. Available: https://www.vi4io.org/_media/io500/about/io500-establishing.pdf.

[22] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, and Michael Houston. Exascale deep learning for climate analytics, 2018.

[23] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[24] Jianwei Li, Wei keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netcdf: A high-performance scientific i/o interface. In *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, pages 39–39, 2003.

[25] Jerry Liu. LlamaIndex, 11 2022.

[26] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023. arXiv:2307.03172.

[27] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment, 2023.

[28] Valentin Liévin, Christoffer Egeberg Hother, Andreas Geert Motzfeldt, and Ole Winther. Can large language models reason about medical questions?, 2023.

[29] Glenn K. Lockwood, Nicholas J. Wright, Shane Snyder, Philip Carns, George Brown, and Kevin Harms. Tokio on clusterstor: Connecting standard tools to enable holistic i/o performance analysis. 2018.

[30] Glenn K. Lockwood, Wucherl Yoo, Suren Byna, Nicholas J. Wright, Shane Snyder, Kevin Harms, Zachary Nault, and Philip Carns. Umami: a recipe for generating meaningful metrics through holistic i/o performance analysis. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, PDSW-DISCS '17, page 55–60, New York, NY, USA, 2017. Association for Computing Machinery.

[31] Jakob Luettgau, Shane Snyder, Tyler Reddy, Nikolaus Awtrey, Kevin Harms, Jean Luca Bez, Rui Wang, Rob Latham, and Philip Carns. Enabling agile analysis of i/o performance data with pydarshan. In *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W '23, page 1380–1391, New York, NY, USA, 2023. Association for Computing Machinery.

[32] Nafiseh Moti, André Brinkmann, Marc-André Vef, Philippe Deniel, Jesus Carretero, Philip Carns, Jean-Thomas Acquaviva, and Reza Salkhordeh. The i/o trace initiative: Building a collaborative i/o archive to advance hpc. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 1216–1222, 2023.

[33] OpenAI. Openai — pricing. https://openai.com/pricing/, 2024.

[34] OpenSFS and EOFS. Lustre file system, 2003-.

[35] Tirthak Patel, Rohan Garg, and Devesh Tiwari. {GIFT}: A coupon based {Throttle-and-Reward} mechanism for fair and efficient {I/O} bandwidth management on parallel storage systems. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 103–119, 2020.

[36] Md Hasanur Rashid, Youbiao He, Forrest Sheng Bao, and Dong Dai. Iopathtune: Adaptive online parameter tuning for parallel file system i/o path. *arXiv preprint arXiv:2301.06622*, 2023.

[37] Juan-Pablo Rivera, Gabriel Mukobi, Anka Reuel, Max Lamparth, Chandler Smith, and Jacquelyn Schneider. Escalation risks from language models in military and diplomatic decision-making, 2024.

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.

[38] Leonard Salewski, Stephan Alaniz, Isabel Rio-Torto, Eric Schulz, and Zeynep Akata. In-context impersonation reveals large language models' strengths and biases, 2023.

[39] Lin Shi, Chiyu Ma, Wenhua Liang, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic investigation of position bias in pairwise comparative assessments by llms, 2024.

[40] Neda Tavakoli, Dong Dai, and Yong Chen. Log-assisted straggler-aware i/o scheduler for high-end computing. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, pages 181–189. IEEE, 2016.

[41] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, and et al. Gemini: A family of highly capable multimodal models, 2024.

[42] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[43] Jeffrey S. Vetter and Michael O. McCracken. Statistical scalability analysis of communication operations in distributed applications. In *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, PPoPP '01, page 123–132, New York, NY, USA, 2001. Association for Computing Machinery.

[44] Chen Wang, Jinghan Sun, Marc Snir, Kathryn Mohror, and Elsa Gonsiorowski. Recorder 2.0: Efficient parallel i/o tracing and analysis. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1–8, 2020.

[45] Teng Wang, Shane Snyder, Glenn Lockwood, Philip Carns, Nicholas Wright, and Suren Byna. Iominer: Large-scale analytics framework for gaining knowledge from i/o logs. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 466–476, 2018.

[46] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

[47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[48] Changrong Xiao, Sean Xin Xu, Kunpeng Zhang, Yufang Wang, and Lei Xia. Evaluating reading comprehension exercises generated by LLMs: A showcase of ChatGPT in education applications. In Ekaterina Kochmar, Jill Burstein, Andrea Horbach, Ronja Laarmann-Quante, Nitin Madnani, Anaïs Tack, Victoria Yaneva, Zheng Yuan, and Torsten Zesch, editors, *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, pages 610–625, Toronto, Canada, July 2023. Association for Computational Linguistics.

[49] Cong Xu, Shane Snyder, Omkar Kulkarni, Vishwanath Venkatesan, Phillip Carns, Surendra Byna, Robert Sisneros, and Kalyana Chadalavada. Dxt: Darshan extended tracing. 1 2019.

[50] Yangming Yue, Yi Lei, Wanghua Shi, and Yi Zhou. Selective propositional reasoning: A cognitive load-aware strategy for enhanced reasoning. In *2023 2nd International Conference on Cloud Computing, Big Data Application and Software Engineering (CBASE)*, pages 231–237, 2023.

[51] Di Zhang, Dong Dai, Runzhou Han, and Mai Zheng. Sentilog: Anomaly detecting on parallel file systems via log-based sentiment analysis. In *Proceedings of the 13th ACM workshop on hot topics in storage and file systems*, pages 86–93, 2021.

[52] Di Zhang, Chris Egersdoerfer, Tabassum Mahmud, Mai Zheng, and Dong Dai. Drill: Log-based anomaly detection for large-scale storage systems using source code analysis. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 189–199. IEEE, 2023.

[53] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B. Hashimoto. Benchmarking large language models for news summarization, 2023.

[54] Weiqun Zhang, Andrew Myers, Kevin Gott, Ann Almgren, and John Bell. Amrex: Block-structured adaptive mesh refinement for multiphysics applications. *The International Journal of High Performance Computing Applications*, 35(6):508–526, 2021.

[55] Wenting Zhao, Tanya Goyal, Yu Ying Chiu, Liwei Jiang, Benjamin Newman, Abhilasha Ravichander, Khyathi Chandu, Ronan Le Bras, Claire Cardie, Yuntian Deng, et al. Wildhallucinations: Evaluating long-form factuality in llms with real-world entity queries. *arXiv preprint arXiv:2407.17468*, 2024.

[56] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

Authorized licensed use limited to: UNIVERSITY OF DELAWARE LIBRARY. Downloaded on October 04,2026 at 21:51:01 UTC from IEEE Xplore. Restrictions apply.