ION: Navigating the HPC I/O Optimization Journey using Large Language Models

Chris Egersdoerfer
cegersdo@charlotte.edu
University of North Carolina at Charlotte
Charlotte, NC, USA

Arnav Sareen
asareen2@charlotte.edu
University of North Carolina at Charlotte
Charlotte, NC, USA

Jean Luca Bez
jlbez@lbl.gov
Lawrence Berkeley National Laboratory
Berkeley, CA, USA

Suren Byna
byna.1@osu.edu
The Ohio State University
Columbus, OH, USA

Dong Dai
ddai@charlotte.edu
University of North Carolina at Charlotte
Charlotte, NC, USA

Abstract

Effectively leveraging the complex software and hardware I/O stacks of HPC systems to deliver needed I/O performance has been a challenging task for domain scientists. To identify and address I/O issues in their applications, scientists largely rely on I/O experts to analyze the recorded I/O traces of their applications and provide insights into the potential issues. However, due to the limited number of I/O experts and the growing demand for data-intensive applications across the wide spectrum of sciences, inaccessibility has become a major bottleneck hindering scientists from maximizing their productivity. Inspired by the recent rapid progress of large language models (LLMs), in this work we propose `IO Navigator` (ION), an LLM-based framework that takes a recorded I/O trace of an application as input and leverages the in-context learning, chain-of-thought, and code generation capabilities of LLMs to comprehensively analyze the I/O trace and provide diagnosis of potential I/O issues. Similar to an I/O expert, ION provides detailed justifications for the diagnosis and an interactive interface for scientists to ask detailed questions about the diagnosis. We illustrate ION's applicability by assessing it on a set of controlled I/O traces generated with different I/O issues. We also demonstrate that ION can match state-of-the-art I/O optimization tools and provide more insightful and adaptive diagnoses for real applications. We believe ION, with its full capabilities, has the potential to become a powerful tool for scientists to navigate through complex I/O subsystems in the future.

1 Introduction

High-performance computing (HPC) systems encapsulate a broad range of scientific applications, such as large-scale climate modeling and forecasting [1, 34] and fluid dynamic simulations [15, 16, 26]. Today, these scientific applications are increasingly data-intensive [9, 31]. Hence, it is of great importance for scientists to effectively leverage the HPC I/O subsystems to deliver optimal I/O performance for their applications. This, however, is a non-trivial task due to the complicated nature of the HPC I/O stack, which includes high-level parallel I/O libraries (e.g., HDF5 [10], PnetCDF [19]), interfaces (e.g., MPI-IO, POSIX, STDIO), and file systems (e.g., Lustre [28], GPFS [12]).

One effective way to help scientists better utilize their I/O systems is to record the I/O traces of their applications for post-hoc analysis and accordingly propose optimization and tuning solutions to improve performance [7, 41]. I/O experts have developed real-time I/O profiling tools such as Darshan [6] and enabled them in modern HPC facilities to provide I/O traces for applications. Based on the recorded traces, tools such as PyDarshan [24] and DXT-Explorer [4] have subsequently been introduced to effectively interpret the traces, identify potential I/O issues, and even provide accurate suggestions to scientists [8, 40].

However, diagnosing I/O issues of applications using their I/O traces has a key issue: it necessitates human I/O experts in the loop. With professionals across the scientific spectrum interested in developing HPC applications, the lack of readily available I/O experts makes diagnosis and optimization of I/O an inaccessible burden and consequently inhibits the potential of their work while costing precious computational time and resources. Hence, an automated tool is urgently needed to make this expertise broadly available to all users.

Recently, substantial strides have been made in the field of large language models (LLMs) such as ChatGPT[1] and Claude[2]. The combination of their anthropomorphic nature and the internet-scale data used to train them makes these models highly approachable and easily applicable to a wide range of tasks, positioning them well to potentially address the inaccessibility of I/O experts. Further, LLMs' in-context learning capability [30] and strong propensity to follow instructions enables them to seamlessly be adapted to several

[1] https://openai.com/chatgpt
[2] https://www.anthropic.com/claude

different domains, allowing the user to confine their generations within the bounds of a predefined set of knowledge, while adjusting these bounds on the fly with new information and guidance. Notable examples of applications leveraging in-context learning include medical questioning [20], military simulation and strategy [29], and educational guidance [38]. Lastly, the robust code interpreters built to work in tandem with state-of-the-art LLMs further provide a swift avenue for running and debugging LLM-generated code for data analysis and visualization [18, 25] if necessary, allowing for fully automated data analysis through multiple iterations of running or debugging code.

These capabilities inspire us to leverage LLMs to interpret, analyze, and reason over HPC applications' I/O traces to guide scientists in optimizing their I/O performance. By doing so, we aim to provide a more accessible tool for domain scientists to navigate through the complexities of their HPC I/O subsystems.

To the end, we propose `I/O Navigator` (ION), an LLM-based framework for analyzing application I/O traces (i.e., Darshan trace). ION is able to: 1) provide a comprehensive diagnosis summary of prevalent I/O issues extracted from Darshan traces, 2) produce a detailed, logical set of analysis steps allowing the user to understand the entire diagnosis process up to the conclusion. ION also provides an interactive interface encouraging scientists to ask questions about the analysis and better understand the conclusion, providing a similar level of intimacy typically limited to only human experts. We demonstrate ION's ability to match state-of-the-art I/O optimization tools and generate detailed diagnosis summaries from I/O traces across scientific domains.

This paper is organized as follows: In §2 we discuss the background of this study and introduce Darshan I/O profiling tool as well as existing efforts in I/O issue diagnosis from recorded traces, particularly Drishti. In §3, we present the key components of ION in detail. We present the extensive experimental results in §4, conclude this paper and discuss the future work in §5.

2 Background

Darshan. Multiple I/O profiling tools, such as STAT [2], mpiP [32], IOPin [13], Recorder [33], and Darshan [6], have been developed to understand applications' I/O behaviors. Among them, Darshan is widely adopted in the HPC community due to its lightweight design and focus on high-level application behavior [27]. Specifically, for each application, Darshan traces key statistical metrics for each file accessed at the I/O-software-stack-level across different types of I/O interfaces including POSIX (Portable Operating System Interface) I/O, MPI (Message Passing Interface) I/O, and Standard I/O. These metrics include the amount of read/write data, aggregate time for read/write/meta operations, ID of the rank issuing I/O requests, and variance of I/O size and time among different application ranks. Darshan also collects Lustre-file-system-level metrics such as stripe width and OST IDs over which a file is striped. Darshan eXtended Tracing (DXT) [39] further extends Darshan by providing a fine-grained record of the application's I/O, such as the file, operation type, offset, length, start and end timestamps, and the issuing rank's ID. Such detailed I/O traces are key in identifying problematic requests and providing guidance. In this study, we focus on Darshan and Darshan DXT logs, which are widely supported across HPC facilities.

Drishti There are a number of tools designed to analyze I/O traces and detect potential I/O problems, such as IOMiner [35], UMAMI [22], TOKIO [21], DXT Explorer [4], recorder-viz [33], and Drishti [3]. The closest one to our work is Drishti, which takes a Darshan trace as input, conducts analysis and reports various (file-based and overall) performance issues as well as actionable tasks for potentially resolving the reported issues. Drishti identifies I/O performance issues based on a set of heuristic-based triggers. Currently, Drishti includes a group of 30 triggers corresponding to various application behaviors and identifies nine different types of I/O issues, such as 'Small I/O Operations', 'Mis-aligned I/O', or 'Imbalanced I/O'.

However, Drishti suffers from several pitfalls that impede its ability to serve as a general I/O expert. First, it relies on pre-defined triggers to identify potential I/O issues. However, setting correct threshold values for these triggers is not a simple task—they may vary significantly among different systems and across distinct workloads. For instance, by default, Drishti considers write requests smaller than 1MB as small writes and reports a 'Small I/O' issue if there are more than 10% small I/Os across all the requests. However, both assumptions (i.e., 1MB and 10%) could be inaccurate, as will be demonstrated in Section 4. Second, Drishti does not provide accurate or contextually based reasoning for identified I/O issues, as its pre-defined triggers and messages make its explanations far-fetched and error-prone. Vitally, missing practical and contextually-based explanations hinder non-expert domain scientists' use of such tools. Finally, Drishti and other existing I/O diagnosis tools do not provide an interactive interface for users to directly ask follow-up questions about their application's I/O analysis and correct their understanding about I/O, which is a key aspect of having a human I/O expert on the team; they can be hardly considered as an automated I/O expert without such a capability. In this study, we create ION with the aim of addressing these issues.

3 Design and Implementation

Figure 1 illustrates the overall workflow of ION, which contains two primary parts. *Extractor* parses the Darshan trace into multiple CSV files for the upcoming analysis. The *Analyzer* then takes charge of constructing multiple prompts

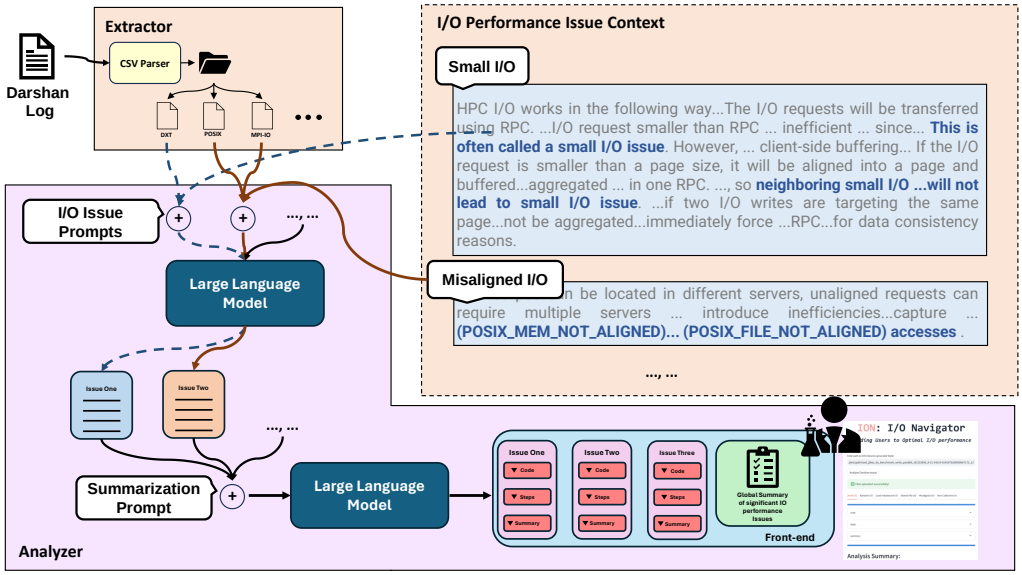

Figure 1. The overall workflow of ION and its key components.

to query the LLM for proper analysis and visually represents the results to the user.

The **ION Extractor** unpacks and parses Darshan log files into the format expected by the later Analyzer. Specifically, it uses Darshan's built-in *darshan-parser* and *darshan-dxt-parser* to read the entirety of each output. In the case of the general *darshan-parser*, each Darshan module which is found within the log file is formatted into a CSV file named after the module itself (i.e., POSIX module data is written to POSIX.csv). Though the exact information contained within each module CSV file is dependent upon the counters that Darshan records for each respective module, the general structure defines each row in the CSV to include a unique file ID and MPI rank combination as well as a column for each of the Darshan module counters. In the case of the *darshan-dxt-parser*, which is only used to extract Darshan's DXT module data, the extractor creates a 'DXT.csv' file which contains a single row for each POSIX or MPI-IO read/write operation. Each row also carries the file ID targeted by the operation, as well as its wall time, offset, and size. The Analyzer will then use these CSV files individually or together to form the actual LLM prompts.

The **ION Analyzer** takes the CSV files created by the aforementioned Extractor as input and automatically constructs multiple prompts based on the *I/O performance Issue Context* to query the LLM.

We construct the *I/O performance Issue Context* to leverage the in-context learning capability of LLMs. From preliminary experimental observations we found that without proper context, LLMs can only generate vacuous and general replies to HPC I/O traces. This is because understanding HPC I/O performance issues is complex and requires in-depth domain knowledge such as a description of when client-side aggregation of small I/O requests is possible or a description of file striping in parallel file systems, which is likely a minimal subset within the training sets of pre-trained LLMs'. Therefore, it is necessary to teach LLMs such detailed knowledge. The benefit of in-context learning over other techniques such as finetuning is that the latter often requires a much higher initial cost and substantial computational cost every time new context is introduced, even if all the context is not pertinent to the particular issue being targeted. In-context learning is incredibly cost-efficient, integrates only what is needed for valid generations, and enables dynamic adjustment of the context to meet the specific needs of scientists.

Initially, in-context learning was attempted by injecting a single large block of text to describe all common I/O performance issues and combining that with a description of the extracted CSV files to query the LLM with a single, voluminous prompt. However, even top-performing LLMs like *gpt-4-1106-preview* faced challenges in extracting key information from the context in this format. In light of this, the ION Analyzer takes a new *divide-and-conquer* approach. Specifically, it creates a set of contexts, each focusing on one particular type of I/O performance issue (i.e., 'Small I/O' or 'Misaligned I/O'). Then, it formats a unique prompt for each type of I/O issue using the corresponding context. Additionally, considering that some issues do not need information from all Darshan modules (i.e. small I/O issues can be identified without considering the MPI-IO level CSV records), a predefined mapping of necessary modules for each issue

type filters the amount of file descriptions included in each prompt's context.

It is worth noting that the *I/O Performance Issue Context* differentiates ION from solely relying on predefined triggers, as was done in Drishti. In each of the contexts, we minimize the use of fixed thresholds, instead describing the nature of the I/O issues and pinpointing key metrics that can be leveraged to identify if the issue is present. Importantly, these metrics are specific system settings such as lustre stripe size which do not require domain expertise to extract unlike the threshold values used by Drishti. Though currently implemented as input hyper-parameters, we consider our future work to include dynamic extraction of these metrics to remove the need for hyper-parameters completely.

In addition to the issue context, each prompt includes a description of the columns in the associated CSV files and an output format description. In light of the noticeable improvements achieved by Chain Of Thought (CoT) Prompting in terms of accuracy, reliability, and explainability [20, 36, 37], we also utilize it to elicit step-by-step reasoning for each diagnosis, leading to better user interpretability and more consistent diagnosis results. Once the prompts for all issues are formatted, they are sent, in parallel, to GPT-4 (*gpt-4-1106-preview*) via the Assistants API. Due to the built-in functionality of the Assistants API, ION can generate diagnosis steps, write/run analysis code, and reason over the results of running generated code, all as part of a single prompt completion.

Once the completions for each prompt are generated, the Analyzer extracts the diagnosis steps, code, and diagnosis conclusion generated by the LLM during each completion. For each issue, these parts are shown in the user interface as represented by the 'Issue *' modals in the *Front-end* section of Figure 1, allowing the user to trace back the diagnosis conclusion by reading the generated analytic code and individual reasoning steps.

The Analyzer also creates a summarization prompt, combining all diagnosis summaries generated by the LLM in the previous step. This is sent to GPT-4 for completion and shown on the user interface upon success. Following the completion of the global diagnosis summary, the Analyzer exposes a message input window that allows the user to ask direct questions about any analysis, reasoning, or result generated throughout any of the diagnosis or summarization steps. This enables users to interact with ION similarly to a human expert through conversation to gain a deeper understanding of the diagnosis output. Further details regarding the content of any prompts used by the ION Analyzer can be found in the following GitHub repository: https://github.com/DIR-LAB/ION.

4 Evaluation

In order to evaluate ION, we used a combination of synthetic traces generated from the IO500 benchmark [14] where I/O issues were manually injected, as well as two real applications experiencing I/O performance issues.

IO500 Results. The IO500 benchmark consists of a comprehensive set of I/O patterns commonly seen throughout HPC applications [14]. By running individual configurations of IO500, such as *ior-hard*, which conducts small, random I/O operations on a shared file or tuning parameters of other sections, such as the transfer size of *ior-easy* to inject misalignment issues, we constructed a set of traces with known ground-truth issues, as shown in Figure 2.

We then applied ION to each of the collected Darshan traces. As shown by the examples, ION successfully identifies each of the known ground truth issues. Further, in many cases ION gives an accurate description of issues that are present in the ground truth but mitigated due to some condition. For example, all IOR-based configurations (*ior-easy*, *ior-hard*, *ior-rnd4k*) conduct I/O operations smaller than the configured Lustre RPC size (4MB), but only *ior-hard*, and *ior-rnd4k* use random access patterns, meaning the negative impact of small operations in these cases is fully realized by the application as opposed to sequential operations, which may be aggregated into larger requests, hence mitigating the overall negative impact on performance. Leveraging the issue context, ION correctly points out this fact in all cases. Similarly, ION can correctly address the presence of a shared file accessed by all ranks but also highlights that during analysis, it found the requests not to overlap, leading to its conclusion that access to the shared file would not lead to stripe conflicts or excessive lock overhead, which are significant risks associated with shared file access.

Real Applications Results. We further examined ION's performance by analyzing traces of two real-world, scientific applications. In the Baseline version, both applications contain I/O issues that experts have previously diagnosed. Their optimized versions, where the primary issues had been fixed, were also analyzed.

The first set of real application traces originates from OpenPMD [11], an open metadata schema representing particle and mesh data from scientific simulations and experiments. In the non-optimized version of this application trace, users noted a significant performance issue caused by a bug in HDF5[5], which caused collective I/O operations to actually create a pattern of individual small and misaligned I/O operations. The optimized version of this trace incorporates a fix to this bug in HDF5, which eliminates the issue of individual small, misaligned operations but introduces some operations with random access patterns. As shown in Figure 3, both Drishti and ION accurately identify the presence of pervasive file misalignment issues and small I/O operations. However, ION found that many of the small I/O

	Ground Truth	ION Outputs
IOR-Easy-2KB-Shared-File	Small read and write requests, but Sequential and Consecutive. 4 ranks read/write to a single shared file. Using Posix api with multiple ranks	...I/O operations ... are **Small and Target the same file**,... **largely sequential**, which **could allow for aggregation and mitigate some inefficiencies**...Significant file **misalignment detected affecting 99.80% of I/O operations**...application is only using POSIX I/O calls and **not employing MPI-IO**, despite the presence of multiple ranks performing I/O...
IOR-Easy-1MB-Shared-File	Small read and write requests, but Sequential and Consecutive. 4 ranks read/write to a single shared file. Using Posix api with multiple ranks	...I/O operations ... **are Smaller than the configured RPC size of 4 MB**... **high aggregation**...hence... **not cause inefficient**... **no overlapping operations within the same stripe**, ... hence **no conflicts or lock overhead at the OSTs**...**0.0% misalignment rate** for a total of 8192 I/O operations...**no utilization of MPI-IO...**
IOR-Easy-1MB-File-per-process	**Small** read and write requests, but Sequential and Consecutive. 4 ranks read/write to there own file. Using posix api with multiple ranks	...**Repetitive small I/O**...significant majority (8184) of these small I/O requests are **potentially aggregatable**... **mitigate the inefficiency**,...each of the four unique files is accessed exclusively by a single rank...**does not exhibit any misaligned I/O behavior**...onsecutive and sequential, suggesting that the application could benefit from the use of MPI-IO's collective and non-blocking operations...**does not use the MPI-IO module**,...
IOR-Hard	**Small read and write request, which cannot be aggregated**. 4 ranks use a shared file. Using Posix api with multiple ranks	...**repetitive pattern of small requests**...**random I/O operations**,... **small I/O**.....there is evidence of **temporal overlap in I/O operations, further indicating contention**...behavior includes a **significant number of misaligned file accesses (799,974)**.....**potential for optimization through parallel I/O libraries like MPI-IO**
IOR-Random-4K-Shared-File	**Small random read/write which cannot be aggregated**. 4 ranks use a shared file. Using Posix api with multiple ranks	... **small and non-sequential...,** there could be a performance concern related to random access patterns...**The POSIX_FILE_NOT_ALIGNED counter indicates 286,442 instances of file misalignment**...approximately 99.61% of the operations..., but it does not use MPI-IO for I/O operation
MD Workbench	**Excessive metadata requests**. Repeated small reads and writes to many files at the same offset.	... **potentially inefficient repetitive small I/O patterns**...application **exhibits high metadata I/O behaviors...could lead to unnecessary load on metadata servers** and potentially create a bottleneck in the system...approximately 51.85% of read/write operations **occur within the same file stripe**...

Figure 2. Evaluation of ION diagnosis output compared to ground truth on IO500 workloads. (note that, these outputs are augmented with color-coding for visualization purposes)

	ION Output	Drishti Output
OpenPMD (Baseline)	Application demonstrates a potentially inefficient I/O pattern...**98.78% of operations being small I/O that might lead to underutilization of RPCs**... **as most small I/O are consecutive, aggregation optimization might be in place**...Significant file misalignment detected...this may contribute to performance degradation due to increased contention... .(..8a_parallel_3Db_0000001.h5) is being accessed by multiple ranks (384 in total)...	issues a high number (275840) of small read requests (i.e., < 1MB).. issues a high number (427386) of small write requests (i.e., < 1MB)...(64.38%) small write requests are to "8a_parallel_3Db_0000001.h5"...Application issues a high number (100.00%) of misaligned file requests...
OpenPMD (Optimized)	The trace file indicates that 3.31% of total I/O operations consist of small-sized operations, **with 88.15% of these exhibiting random access patterns... However, the random access operation count per rank and the total volume of data transferred through these patterns are low**...not affecting the entire application's I/O performance.	Application is issuing a high number (565) of random read operations (35.25%)...
E2E (Baseline)	**misaligned memory accesses...(approximately 99.8%) of the write operations are misaligned**... ...**Rank 0 also has much larger summed I/O sizes ..., which indicates that rank 0 is doing much more work**...	Application issues a high number (99.81%) of misaligned file requests...Load imbalance of 99.90% detected while accessing "3d_32_32_16_32_32_32.nc4"...
E2E (Optimized)	**...a pervasive issue with file access alignment, with 99.8% of file I/O operations being misaligned.** ... **A subset of 64 out of the 1024 ranks exhibit a significantly higher number of I/O operations per second...their throughput stats far exceeding one standard deviation above the mean...these ranks contribute to approximately 98.23% of the total write operations...it is worth investigating further to determine if this behavior is intentional (e.g., based on the application algorithm) or if it can be optimized for better load distribution.**	Application issues a high number (99.80%) of misaligned file requests...

Figure 3. Comparison of ION and Drishti Diagnosis for real applications (note that, outputs are augmented with color-coding for visualization purposes)

operations are consecutive and, therefore, accurately points out the potential for aggregation, which would minimize the possible negative impact of small operations on the file system. In the optimized version, both ION and Drishti accurately identify the presence of operations with random access patterns; however, ION accurately puts the number of random accesses into the context of the number of ranks conducting I/O and further elucidates upon the amount of data processed by random operations.

The second set of real application traces originates from the end-to-end (E2E)[23] domain decomposition I/O kernel. In the non-optimized version of this application, an overwhelming load imbalance on rank 0 was caused by the use of fill values for subsequently overwritten datasets. As noted by the users, disabling this behavior created a 10× speedup[3]. In the baseline version of this application, both ION and Drishti accurately identify the presence of pervasive file misalignment and load imbalance, but ION provides more useful detail regarding the rank responsible for the perceived load imbalance and which is doing more work in terms of throughput and operation count. In the optimized version of this application, both Drishti and ION continue to identify file misalignment, but ION recognizes that while the load

imbalance is no longer imbalanced solely on rank 0, there seems to be a subset of ranks doing more work than the rest, indicating that this may be inherent to the algorithm rather than warning the user of a significant issue.

5 Conclusion and Future Work

In this study, we propose and implement ION, an LLM-based framework to analyze HPC applications' I/O traces and provide diagnosis regarding potential I/O performance issues. ION leverages in-context learning, chain-of-thought, and code analysis capabilities of LLMs (i.e. GPT4) to comprehensively understand the I/O traces and accurately diagnose I/O issues for scientists. We show that, without relying on extensive pre-defined triggers or thresholds which require extensive expertise to accurately set, ION can match or exceed the performance of state-of-the-art I/O diagnosis tools (i.e., Drishti). Currently, ION is still a proof-of-concept, but with its promising results, we plan to 1) build more comprehensive knowledge base about HPC I/O so that the underlying LLM can conduct more accurate reasoning; 2) optimize the prompts to enable consistency checking of the diagnosis results; 3) test alternatives to in-context learning like Retrieval-Augmented Generation (RAG) [17] for more efficient continued interactive interfaces.

Acknowledgments

We sincerely thank the anonymous reviewers for their valuable feedback. This work was supported in part by NSF grants CNS-2008265 and CCF-2412345. This research 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 at Lawrence Berkeley National Laboratory (LBNL) and under a subcontract at The Ohio State University under a subcontract (GR130493).

References

[1] Jean-Claude André, Giovanni Aloisio, Joachim Biercamp, Reinhard Budich, Sylvie Joussaume, Bryan Lawrence, and Sophie Valcke. 2014. High-Performance Computing for Climate Modeling. *Bulletin of the American Meteorological Society* 95, 5 (2014), ES97 – ES100. https://doi.org/10.1175/BAMS-D-13-00098.1

[2] Dorian C. Arnold, Dong H. Ahn, Bronis R. de Supinski, Gregory L. Lee, Barton P. Miller, and Martin Schulz. 2007. Stack Trace Analysis for Large Scale Debugging. In *2007 IEEE International Parallel and Distributed Processing Symposium*. 1–10. https://doi.org/10.1109/IPDPS.2007.370254

[3] Jean Luca Bez, Hammad Ather, and Suren Byna. 2022. Drishti: Guiding End-Users in the I/O Optimization Journey. In *2022 IEEE/ACM International Parallel Data Systems Workshop (PDSW)*. 1–6. https://doi.org/10.1109/PDSW56643.2022.00006

[4] Jean Luca Bez, Houjun Tang, Bing Xie, David Williams-Young, Rob Latham, Rob Ross, Sarp Oral, and Suren Byna. 2021. I/O Bottleneck Detection and Tuning: Connecting the Dots using Interactive Log Analysis. In *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)*. 15–22. https://doi.org/10.1109/PDSW54622.2021.00008

[5] Jean Luca Bez, Houjun Tang, Bing Xie, David Williams-Young, Rob Latham, Rob Ross, Sarp Oral, and Suren Byna. 2021. I/O Bottleneck Detection and Tuning: Connecting the Dots using Interactive Log Analysis. In *2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW)*. 15–22. https://doi.org/10.1109/PDSW54622.2021.00008

[6] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and Improving Computational Science Storage Access through Continuous Characterization. *ACM Trans. Storage* 7, 3, Article 8 (oct 2011), 26 pages. https://doi.org/10.1145/2027066.2027068

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

[8] Chris Egersdoerfer, Di Zhang, and Dong Dai. 2023. 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*. 315–316.

[9] Girone, Maria, Southwick, David, Khristenko, Viktor, Medeiros, Miguel F., Giordano, Domenico, Brevik Høgstøyl, Ingvild, and Atzori, Luca. 2021. Exploitation of HPC Resources for data intensive sciences. *EPJ Web Conf.* 251 (2021), 02042. https://doi.org/10.1051/epjconf/202125102042

[10] The HDF Group. 1997-. *The HDF5® Library & File Format*. https://www.hdfgroup.org/solutions/hdf5/

[11] Axel Huebl, Rémi Lehe, Jean-Luc Vay, David P. Grote, Ivo Sbalzarini, Stephan Kuschel, David Sagan, Frédéric Pérez, Fabian Koller, and Michael Bussmann. 2018. *openPMD 1.1.0: Base paths for mesh- and particle- only files and updated attributes*. https://doi.org/10.5281/zenodo.1167843

[12] IBM. 1998-. *General Parallel File System 4.1.0.4*. https://www.ibm.com/docs/en/gpfs

[13] Seong Jo Kim, Seung Woo Son, Wei-keng Liao, Mahmut Kandemir, Rajeev Thakur, and Alok Choudhary. 2012. IOPin: Runtime Profiling of Parallel I/O in HPC Systems. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. 18–23. https://doi.org/10.1109/SC.Companion.2012.14

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

[15] Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, and Michael Houston. 2018. Exascale Deep Learning for Climate Analytics. arXiv:1810.01993 [cs.DC]

[16] Marius Kurz, Philipp Offenhäuser, Dominic Viola, Oleksandr Shcherbakov, Michael Resch, and Andrea Beck. 2022. Deep reinforcement learning for computational fluid dynamics on HPC systems. *Journal of Computational Science* 65 (2022), 101884. https://doi.org/10.1016/j.jocs.2022.101884

[17] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL]

[18] Jinyang Li, Nan Huo, Yan Gao, Jiayi Shi, Yingxiu Zhao, Ge Qu, Yurong Wu, Chenhao Ma, Jian-Guang Lou, and Reynold Cheng. 2024. Tapilot-Crossing: Benchmarking and Evolving LLMs Towards Interactive Data Analysis Agents. arXiv:2403.05307 [cs.AI]

[19] Jianwei Li, Wei keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. 2003. Parallel netCDF: A High-Performance Scientific I/O Interface. In *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. 39–39. https://doi.org/10.1109/SC.2003.10053

[20] Valentin Liévin, Christoffer Egeberg Hother, Andreas Geert Motzfeldt, and Ole Winther. 2023. Can large language models reason about medical questions? arXiv:2207.08143 [cs.CL]

[21] Glenn K. Lockwood, Nicholas J. Wright, Shane Snyder, Philip Carns, George Brown, and Kevin Harms. 2018. TOKIO on ClusterStor: Connecting Standard Tools to Enable Holistic I/O Performance Analysis. (2018). https://www.osti.gov/biblio/1632125

[22] Glenn K. Lockwood, Wucherl Yoo, Suren Byna, Nicholas J. Wright, Shane Snyder, Kevin Harms, Zachary Nault, and Philip Carns. 2017. 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* (Denver, Colorado) *(PDSW-DISCS '17)*. Association for Computing Machinery, New York, NY, USA, 55–60. https://doi.org/10.1145/3149393.3149395

[23] Jay Lofstead, Milo Polte, Garth Gibson, Scott Klasky, Karsten Schwan, Ron Oldfield, Matthew Wolf, and Qing Liu. 2011. Six degrees of scientific data: reading patterns for extreme scale science IO. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing* (San Jose, California, USA) *(HPDC '11)*. Association for Computing Machinery, New York, NY, USA, 49–60. https://doi.org/10.1145/1996130.1996139

[24] Jakob Luettgau, Shane Snyder, Tyler Reddy, Nikolaus Awtrey, Kevin Harms, Jean Luca Bez, Rui Wang, Rob Latham, and Philip Carns. 2023. 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)*. Association for Computing Machinery, New York, NY, USA, 1380–1391. https://doi.org/10.1145/3624062.3624207

[25] Paula Maddigan and Teo Susnjak. 2023. Chat2VIS: Generating Data Visualisations via Natural Language using ChatGPT, Codex and GPT-3 Large Language Models. arXiv:2302.02094 [cs.HC]

[26] Peter Mora, Gabriele Morra, and David A Yuen. 2019. A concise python implementation of the lattice Boltzmann method on HPC for geo-fluid flow. *Geophysical Journal International* 220, 1 (2019), 682–702. https://doi.org/10.1093/gji/ggz423

[27] Nafiseh Moti, André Brinkmann, Marc-André Vef, Philippe Deniel, Jesus Carretero, Philip Carns, Jean-Thomas Acquaviva, and Reza Salkhordeh. 2023. 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*. 1216–1222.

[28] OpenSFS and EOFS. 2003-. *Lustre File System*. https://www.lustre.org/

[29] Juan-Pablo Rivera, Gabriel Mukobi, Anka Reuel, Max Lamparth, Chandler Smith, and Jacquelyn Schneider. 2024. Escalation Risks from Language Models in Military and Diplomatic Decision-Making. arXiv:2401.03408 [cs.AI]

[30] Leonard Salewski, Stephan Alaniz, Isabel Rio-Torto, Eric Schulz, and Zeynep Akata. 2023. In-Context Impersonation Reveals Large Language Models' Strengths and Biases. arXiv:2305.14930 [cs.AI]

[31] Amoghavarsha Suresh, Pietro Cicotti, and Laura Carrington. 2014. Evaluation of emerging memory technologies for HPC, data intensive applications. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. 239–247. https://doi.org/10.1109/CLUSTER.2014.6968745

[32] Jeffrey S. Vetter and Michael O. McCracken. 2001. Statistical scalability analysis of communication operations in distributed applications. In *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming* (Snowbird, Utah, USA) *(PPoPP '01)*. Association for Computing Machinery, New York, NY, USA, 123–132. https://doi.org/10.1145/379539.379590

[33] Chen Wang, Jinghan Sun, Marc Snir, Kathryn Mohror, and Elsa Gonsiorowski. 2020. Recorder 2.0: Efficient Parallel I/O Tracing and Analysis. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1–8. https://doi.org/10.1109/IPDPSW50202.2020.00176

[34] Feiyi Wang, John Harney, Galen Shipman, Dean Williams, and Luca Cinquini. 2011. Building a large scale climate data system in support of HPC environment. In *2011 7th International Conference on Next Generation Web Services Practices*. 380–385. https://doi.org/10.1109/NWeSP.2011.6088209

[35] Teng Wang, Shane Snyder, Glenn Lockwood, Philip Carns, Nicholas Wright, and Suren Byna. 2018. IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 466–476. https://doi.org/10.1109/CLUSTER.2018.00062

[36] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. arXiv:2203.11171 [cs.CL]

[37] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 24824–24837. https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[38] Changrong Xiao, Sean Xin Xu, Kunpeng Zhang, Yufang Wang, and Lei Xia. 2023. Evaluating Reading Comprehension Exercises Generated by LLMs: A Showcase of ChatGPT in Education Applications. In *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, Ekaterina Kochmar, Jill Burstein, Andrea Horbach, Ronja Laarmann-Quante, Nitin Madnani, Anaïs Tack, Victoria Yaneva, Zheng Yuan, and Torsten Zesch (Eds.). Association for Computational Linguistics, Toronto, Canada, 610–625. https://doi.org/10.18653/v1/2023.bea-1.52

[39] Cong Xu, Shane Snyder, Omkar Kulkarni, Vishwanath Venkatesan, Phillip Carns, Surendra Byna, Robert Sisneros, and Kalyana Chadalavada. 2019. DXT: Darshan eXtended Tracing. (1 2019). https://www.osti.gov/biblio/1490709

[40] Di Zhang, Dong Dai, Runzhou Han, and Mai Zheng. 2021. 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*. 86–93.

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