Augmenting LLMs for HPC I/O Performance Diagnosis

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

[1] University of Delaware, [2] University of North Carolina art Charlotte, [3] Lawrence Berkeley National Lab, [4] The Ohio State. University, [5] North Carolina State University

Introduction

- The High Performance Computing (HPC) I/O stack consists of many complex and interdependent layers
- As demand from domain scientists to build data-intensive applications grows, the small supply of I/O experts to provide guidance is highlighted and accessibility to expert knowledge is blocked
- In this work, we explore the use of LLMs to fill this knowledge accessibility gap and implement key designs to avoid their baseline shortcomings
- We additionally construct a dataset of application I/O trace logs with a set of ground truth labels representing common performance issues exhibited by each

Background

I/O Performance Diagnosis
- Numerous tools allow for detailed, offline I/O behavior inspection by generating trace logs (i.e. Darshan, Recorder)
- Trace analysis tools provide APIs to extract trace-log content (PyDarshan, DXT-Explorer)
- Such analysis tools still require expert-knowledge to lead to useful diagnosis insights and recommendations

Large Language Models (Strengths)
- Provide an interpretable interface for users
- Improved instruction-following and in-context-learning make them viable as domain-specific assistants

Large Language Models (Shortcomings)
- Bounded context length; trace log size is not bounded
- HPC I/O domain knowledge is limited
- Propensity to hallucinate domain-specific info

Terminology

- **Darshan:** Tool used to collect HPC application I/O trace logs
- **RAG:** Retrieval Augmented Generation uses retrieved sources to augment LLM generations
- **IO500:** Configurable I/O benchmark consisting of common HPC I/O patterns run as separate tasks
- **LlamaIndex:** An open-source framework for creating an embedding index and efficient retrieval

Design & Implementation

Module Preprocessor
- Splits Darshan trace logs into a separate file per module (i.e. POSIX, MPI-IO)
- Implements summary methods to extract key information from each module file

Domain Knowledge Index
- We surveyed the last 5 years of I/O related publications
- Selected 66 key works with relevant information
- Chunked, embedded and stored content in a search index using LlamaIndex

Initial Diagnosis Generation
1. Search the vector index for relevant information to summary function output
2. Prompt LLM to generate a diagnosis of any issues based on summary function output and retrieved sources. This step is commonly known as RAG

Pairwise Merge
- Initial generated diagnoses are merged 2 at a time
- Minimizes the possibility for LLMs to forget important context when merging multiple summaries at once
- Allows for the single final diagnosis to capture all relevant and important information

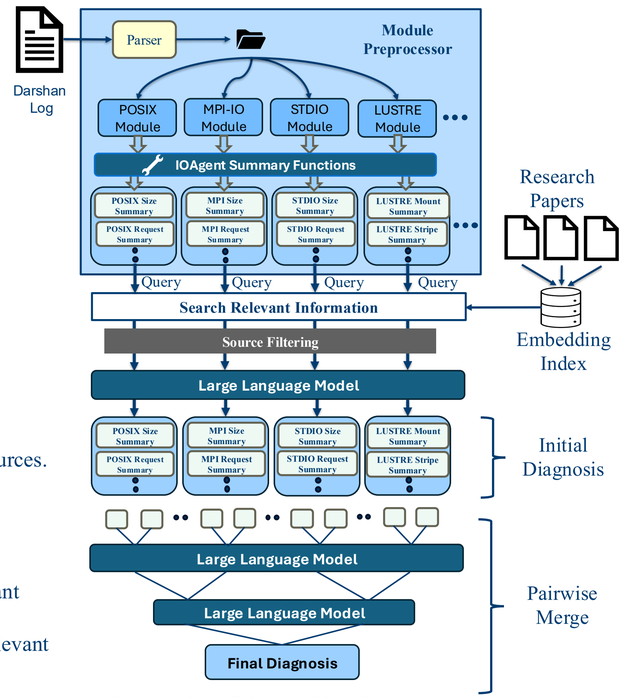

Figure 1: Overall design of the I/O Navigator

Post-Diagnosis Interaction

- The language-centered design of the I/O Navigator allows for rich and insightful user interaction after the final diagnosis is generated
- A real example is presented in *Figure 2* showing a user inquiring about the ideal stripe size setting for their application. The LLM effectively uses the final diagnosis content to recommend 4 and 4MB for stripe count and size with correct code samples

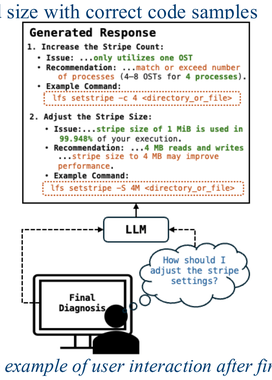

Figure 2: An example of user interaction after final diagnosis

Trace Dataset

- We compiled a dataset of trace logs from the following three sources:
  - **Simple Benchmark:** 10 traces generated using simple C source codes to exhibit at least 1 of the labelled issues ("SB" column in *Figure 3*)
  - **IO500:** 21 traces generated using various configurations of the popular IO500 benchmark to intentionally exhibit sets of labelled issues ("IO500" column in *Figure 3*)
  - **Real Applications:** 9 traces from real applications run on production systems, labelled based on manual analysis ("RA" column in *Figure 3*)

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

Figure 3: Table of classified I/O performance issues in the dataset

Evaluation

Evaluation Setup
- We use an LLM to compare the output of our approach with the outputs of existing approaches, ION and Drishti
- ION is a LLM based diagnosis tool which does not use an embedding index or module-based preprocessor
- Drishti is an expert–guided tool which allows expert users to tune threshold values
- We evaluate our design on three criteria:
  - **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
  - **Interpretability:** evaluate how readable and understandable the provided information is for users at any level of familiarity with HPC I/O

Results
- *Figure 4* shows the ranking results on all ranking criteria and dataset sources.
- We show that our approach achieves higher average rankings in all cases using a frontier model (gpt-4o) as a backbone and maintains similar scores using an open source model as a backbone (Llama-3.2-70B)

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
	Our Approach (gpt-4o)	**0.630**	**0.655**	**0.620**	**0.641**
	Our Approach (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
	Our Approach (gpt-4o)	0.565	**0.615**	**0.639**	**0.609**
	Our Approach (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
	Our Approach (gpt-4o)	0.546	**0.659**	**0.713**	**0.645**
	Our Approach (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
	Our Approach (gpt-4o)	0.580	**0.643**	**0.657**	**0.632**
	Our Approach (llama-3.1-70B)	**0.670**	0.520	0.500	0.550

Figure 4: Table of ranking results on evaluated criteria and datasets

Conclusion

- We present a novel LLM-based framework for I/O performance analysis of HPC applications which achieves superior accuracy as well as higher rankings on qualitative criteria compared to existing tools
- We also show how the proposed approach enables rich user interaction, as well as highly personalized recommendations for I/O optimization.
- Finally, we show our approach is model-agnostic, allowing for high quality diagnosis even with open source LLMs small enough to run locally
- As our future work, we plan to further advance the post-diagnosis user-interaction flow to allow farther reaching recommendations (i.e. application source code changes)