

S3LLM: Large-Scale Scientific Software Understanding with *LLMs* Using Source, Metadata, and Document

Kareem Shaik¹, Dali Wang², Weijian Zheng³, Qinglei Cao⁴,
Heng Fan¹, Peter Schwartz², and Yunhe Feng¹

¹ University of North Texas, Denton, TX 76207, USA

kareembabashaik@my.unt.edu, {heng.fan, yunhe.feng}@unt.edu

² Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA

{wangd, schwartzpd}@ornl.gov

³ Argonne National Laboratory, Lemont, IL 60439, USA wzheng@anl.gov

⁴ Saint Louis University, St. Louis, MO 63103, USA qinglei.cao@slu.edu

Abstract. The understanding of large-scale scientific software is a significant challenge due to its diverse codebase, extensive code length, and target computing architectures. The emergence of generative AI, specifically large language models (LLMs), provides novel pathways for understanding such complex scientific codes. This paper presents *S3LLM*, an LLM-based framework designed to enable the examination of source code, code metadata, and summarized information in conjunction with textual technical reports in an interactive, conversational manner through a user-friendly interface. *S3LLM* leverages open-source LLaMA-2 models to enhance code analysis through the automatic transformation of natural language queries into domain-specific language (DSL) queries. In addition, *S3LLM* is equipped to handle diverse metadata types, including DOT, SQL, and customized formats. Furthermore, *S3LLM* incorporates retrieval-augmented generation (RAG) and LangChain technologies to directly query extensive documents. *S3LLM* demonstrates the potential of using locally deployed open-source LLMs for the rapid understanding of large-scale scientific computing software, eliminating the need for extensive coding expertise and thereby making the process more efficient and effective. *S3LLM* is available at <https://github.com/ResponsibleAILab/s3llm>.

Keywords: Large-Scale Scientific Software, Research Software Analysis, E3SM Land Model, Retrieval Augmented Generation (RAG), LLM, LLaMA, ChatGPT

This manuscript has been authored by UT-Battelle, LLC under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results

1 Introduction

Large-scale scientific computing software is crucial in various scientific fields, undergoing extensive development cycles that lead to the formation of intricate software libraries and ecosystems. This complexity stems from the lengthy development periods, ongoing extensions, and evolving development paradigms, making it imperative to provide users with insights into these computing tools. However, understanding such software is a challenging task for several reasons. First, large-scale scientific software often incorporates multiple programming languages, including older languages such as Fortran and Pascal, which poses a significant challenge for contemporary programmers trying to understand the code. Second, the large volume of scientific software, which may encompass millions of lines of code, makes comprehensively understanding each segment of the code an additional obstacle. Lastly, the documentation for these software systems is sometimes less than ideal, often lacking detailed explanations, which further complicates the task of gaining a thorough understanding of the software.

To enhance comprehension of large-scale scientific software, numerous tools have been devised to aid in code analysis and documentation. For instance, Doxygen [1] is capable of generating documentation from the source code and performing static code analysis for software source trees. Nonetheless, the available tools are primarily tailored for static code analysis and lack the capability to accommodate dynamic queries. Moreover, given the complexity inherent in large-scale scientific software, both developers and users may struggle to formulate queries in both instructed (e.g., textual documents) and structured formats (e.g., SQL). Therefore, it is imperative to devise methods for understanding and parsing large-scale scientific software that are both user-friendly and precise.

The emergence of generative AI, particularly large language models (LLMs), represents a new era in software comprehension and interaction. LLMs have shown remarkable capabilities across various tasks, including chatbot interactions [15, 3, 5], text summarization [13, 4, 14], and content creation [2, 7, 8], demonstrating their potential to vastly improve programming and documentation practices. Beyond these applications, LLMs offer promising solutions for understanding the complex landscapes of large-scale scientific software [12]. By leveraging LLMs, we can envision a future in which software comprehension is not only more accessible but also more intuitive, enabling users to query and interact with software in natural language. This paper introduces *S3LLM*, a novel framework that embodies this vision by providing a user-friendly interface for interacting with scientific computing software through natural language queries. *S3LLM* aims to bridge the gap between the world of complex scientific software and the diverse community of users and developers, fostering a deeper understanding and facilitating more effective use of these critical computational tools.

As opposed to most existing works on software understanding, the proposed *S3LLM* can handle various types of tasks for large-scale scientific software

of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

understanding—including source code query, metadata analysis, and text-based technical report understanding. *S3LLM* is capable of conducting queries over the information extracted from source code in diverse formats, such as DOT (graph description language)⁵ and relational databases. By leveraging the few-shot learning capability of LLMs, *S3LLM* can also generate domain-specific language (DSL) queries, such as Feature Query Language (FQL) [17], to gather and extract software features through code analysis. Furthermore, *S3LLM* implements LangChain and retrieval-augmented generation (RAG) [9] schemes to enable text-based queries from technical reports and project summaries. More importantly, all the aforementioned interactions and inquiries facilitated by *S3LLM* are executed utilizing natural language.

The contributions outlined in this paper are summarized as follows:

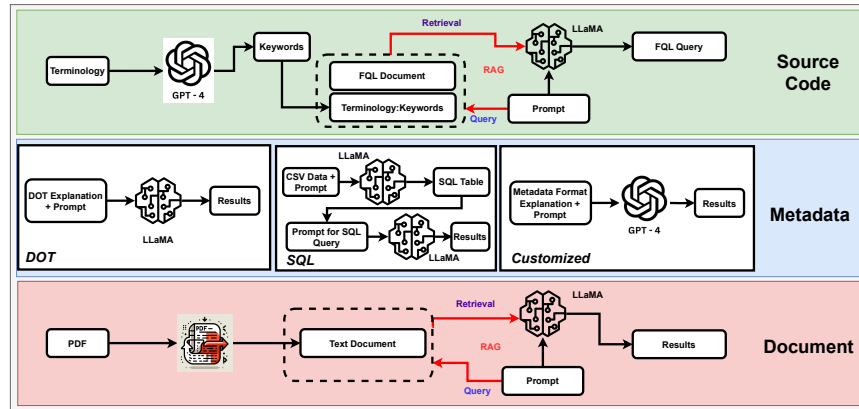
- We conceptualized, designed, and implemented *S3LLM*, a novel framework that utilizes LLMs to improve the understanding of large-scale scientific software. This framework excels in analyzing source code, metadata, and textual technical reports, providing a holistic approach to software comprehension.
- *S3LLM* presents a user-friendly natural language-based interface that allows users, even those with limited programming knowledge, to easily query and gain insights into scientific software.
- Given the need to balance inference speed with the framework’s computational demands, *S3LLM* provides three options featuring LLaMA-2 models with 7B, 13B, and 70B parameters, allowing users to choose the most appropriate model based on their specific requirements.
- Tests on the Energy Exascale Earth System Model (E3SM) [6] shows our model’s effectiveness in analyzing source code, metadata, and documents.
- We contribute to the wider scientific computing community by releasing *S3LLM* as an open-source tool, ensuring broad accessibility and usefulness across a spectrum of scientific computing applications and research pursuits.

2 Method

2.1 Framework Overview

Figure 1 shows that *S3LLM* consists of three main components designed to process source code, code metadata, and textual technical documents. Central to all three components are LLaMA-2 models, which translate natural language into domain-specific language (DSL) queries or perform in-text information analysis and retrieval. Some components within *S3LLM* use the RAG methodology to improve LLM responses by referencing an external knowledge base, enhancing the model’s output. *S3LLM* employs open-source LLaMA-2 models (7B, 13B, and 70B) for language understanding, providing a range of model sizes and ensuring reproducibility across different computational settings.

⁵ [https://en.wikipedia.org/wiki/DOT_\(graph_description_language\)](https://en.wikipedia.org/wiki/DOT_(graph_description_language))

Fig. 1: Framework overview of *S3LLM*

2.2 Source Code Analysis

Large-scale scientific software like E3SM often involves over a million lines of source code, posing challenges for open-source LLMs due to the massive token count exceeding their context windows. Additionally, contemporary sophisticated source code analysis tools demand extensive programming knowledge or specialized domain expertise from users. To tackle these issues, *S3LLM* integrates the analytical strengths of existing tools with the natural language capabilities of LLMs, enabling source code queries in natural language without loading the entire codebase into the LLMs.

To demonstrate *S3LLM*'s strategy for LLM-based source code analysis, we use XScan as a key backend engine [16]. XScan, an integrated toolkit, extracts essential data from scientific code like lines of code, programming languages, library dependencies, and features of parallel software. While XScan simplifies tasks like line tallying, it employs FQL for deeper static code analysis. FQL, which handles queries like Library Utilization, Version Assessment, and Feature Enumeration, requires skills for precise query crafting. *S3LLM* bridges the gap by enabling accessible source code analysis with less need for extensive programming knowledge. Additionally, the core LLMs of *S3LLM* are versatile and can integrate with various source code analysis engines, not limited to XScan.

In transforming natural language into FQL queries, *S3LLM* focuses on three aspects: understanding FQL's purpose and syntax, grasping relevant terminologies like high-performance computing standards, and accurately translating natural language to FQL. To equip LLM models with a solid understanding of FQL, we integrated a foundational FQL document and query examples into the RAG framework as external resources. Addressing the challenge of converting specific terminologies into programming code keywords, initial efforts using open-source LLMs were inadequate. Consequently, GPT-4 was chosen for its ability to translate terminology within *S3LLM*. The model uses few-shot learning and RAG-enhanced contexts to generate FQL queries from natural language inputs.

2.3 Software Metadata Comprehension

To enhance S3LLM’s handling of diverse scientific computing software information, we developed a dual-phase strategy. Initially, we enriched the LLMs’ understanding of metadata formats by embedding detailed context using the RAG technique or direct integration into prompts. S3LLM then allows metadata interrogation through natural language prompts. It features two prompting modes: zero-shot, which lets users query metadata without prior examples, and few-shot, which uses example inputs to refine responses, proving more effective for complex metadata. S3LLM supports three metadata types from scientific software: DOT, SQL, and third-party specified formats, with the flexibility to expand to more in the future. For DOT, we enhance understanding by including a detailed format explanation in prompts. SQL queries are directly generated using LLaMA-2 models, bypassing traditional instruction-based prompts. For custom third-party formats, we showcased S3LLM’s capabilities with SPEL, a toolkit for adapting E3SM models for GPU execution, demonstrating S3LLM’s ability to handle specialized data formats through GPT-4. This broadens its utility across various metadata types.

2.4 Technical Document Interpretation

Large-scale scientific software is often accompanied by detailed documentation such as technical reports, user manuals, and research papers, which can be challenging to navigate. To efficiently extract information from these extensive texts, *S3LLM* utilizes a combination of RAG and LLMs to improve the accuracy of document-related queries. The RAG framework operates in three phases: document indexing, retrieval, and generation. It first segments external texts for the LLM’s contextual analysis, then indexes and stores document embeddings. Upon receiving a query, it retrieves relevant embeddings to provide a context window for the LLM, which generates responses using both the query and the retrieved data. *S3LLM* incorporates the LangChain framework, an advanced tool for developing LLM applications, to implement RAG. LangChain’s DocumentLoaders and Text Splitters organize and segment documents for query processing. VectorStore and Embeddings models, using *all-MiniLM-L6-v2* for embeddings and a FAISS-based index for efficient retrieval, maintain document embeddings. The Retriever component fetches relevant segments for inclusion in user prompts. Finally, the refined query, enhanced with retrieved data, is processed by LLaMA-2 to produce customized responses, demonstrating the effective integration of RAG to enhance document comprehension in scientific software.

3 Case Study

We deployed *S3LLM* on E3SM as a case study to demonstrate its effectiveness in analyzing source code, code metadata, and text-based technical reports. E3SM is a state-of-the-art Earth system modeling framework, developed and supported by the United States Department of Energy (DOE)[6].

3.1 Source Code Query Results

To illustrate the effectiveness of *S3LLM* in source code analysis, we assessed its ability to generate FQL queries across three distinct categories: Library Utilization Queries, Version Assessment Queries, and Feature Enumeration Queries. Through the use of crafted prompts, as demonstrated in the subsequent text box, *S3LLM* successfully generates the correct FQL query in certain instances. Specifically, for the Library Utilization Query, *S3LLM* produces the precise FQL query in a zero-shot manner. This accuracy is likely attributed to the pretraining of the LLaMA-2 model within *S3LLM* on RAG supporting documents that include FQL queries and HPC terminologies. Similarly, for Version Assessment and Feature Enumeration Query, “*Please Generate FQL query to Find the minimum version of MPI*” and “*Please Generate FQL query to List MPI process topology used in the Code*” can prompt *S3LLM* to generate accurate queries.

Prompt for Library Utilization Query:
Please Generate FQL query to Check Whether OpenMP is used

Response:
 CHECK (omp task || end task || omp taskloop || omp taskloop simd
 || omp taskyield) WHERE (*)

3.2 Metadata Comprehension with *S3LLM*

In our study, we utilized three distinct forms of metadata to demonstrate the capabilities of *S3LLM* in enhancing metadata comprehension. These include DOT files produced by XScan [16], CSV files generated by SPEL [11]⁶, and bespoke metadata formats also created by SPEL, all derived from E3SM.

First, *S3LLM*’s efficacy was assessed on standardized DOT files, as produced by XScan and on a unique metadata format devised by SPEL. A DOT file, as illustrated in the box below, represents a directed graph that visualizes the interactions between caller and callee functions within E3SM, offering insights into the software’s structural dynamics.

Prompt for Summarizing DOT File:
DOT files can describe a directed graph using the keyword of “digraph”. Inside a directed graph, each line starts with a module name, which is followed by the “:” symbol. For example, “allocationmod” is the module in “allocationmod::allocationinit -> clm_time_manager.” Please list all the unique modules in the following DOT file describing directed graph G.

```
digraph G {
  emi_datamod::emidcopy -> emi_datamod::emidalloctememory_real_3d
  ...
  firemod::firefluxes -> histfilemod::hist_addfldld
}
```

Response:
 Sure! Here are the unique modules in the directed graph G described by the DOT file:
 1. emi_datamod ... 17. verticalprofilemod There are 17 unique modules in total.

⁶ A software toolkit for porting E3SM land models onto GPUs using OpenACC.

SQL is widely utilized for managing data in relational databases, and language models can translate natural language into SQL queries efficiently. We demonstrated the SQL capabilities of *S3LLM* using two CSV files from SPEL [11]. The process involved two steps. Initially, *S3LLM* generated SQL statements to convert the subtables into SQL tables. Then, it created specific SQL queries. Our experimental results demonstrate *S3LLM*'s skill in handling and manipulating SQL tables for complex database tasks.

Additionally, a sequence of executions for `LakeTemperature` as delineated by SPEL [11] has been tested. The result offers an in-depth view of variable usage during the execution process, which is beneficial for code performance optimization (through asynchronous kernel launch) on GPUs. This structured approach to metadata analysis underscores *S3LLM*'s capability to navigate and elucidate complex metadata representations, enhancing comprehension and facilitating deeper insights into the software's operational mechanisms. Due to the space limitation, please refer to our GitHub repository for more details.

3.3 Document Parsing

This section demonstrates the effectiveness of combining the RAG framework and LangChain with LLMs, as described in Section 2.4, for extracting specific technical information from comprehensive scientific documents. We utilized a 28-page technical report on lake temperature (in PDF format), from the Community Land Model [10], as the testbed. The example experiment in the following colored box confirms *S3LLM*'s proficiency in extracting complex physical processes, such as phase changes, and its capability in both comprehension and explanation.

Prompt for Querying Technical Report of Lake Temperature:

What is the significance of phase changes in the model, and how are they accounted for?

Response: Phase changes, such as freezing and melting, play a significant role in the atmosphere model, as they affect the energy balance at the land surface. The model accounts for phase changes by calculating heat diffusion and phase change in given layers based on temperature and moisture conditions. The available energy for melting is computed as function of temperature and moisture content.

4 Conclusion

This paper introduces *S3LLM*, a robust framework built on LLMs designed to understand the intricacies of large-scale scientific software. Through advanced techniques such as instruction-based prompting, integration with external queries, and the use of RAG and LangChain, *S3LLM* enhances the operational capabilities of pre-trained LLMs, and its effectiveness across various data formats was demonstrated herein. This framework not only underscores our current achievements but also lights the way for future advancements in generative AI and software engineering, particularly within scientific computing.

References

1. Doxygen. <https://www.doxygen.nl/>. Accessed: 2024-03-01.
2. Arkadeep Acharya, Brijraj Singh, and Naoyuki Onoe. Llm based generation of item-description for recommendation system. In *Proceedings of the 17th ACM RecSys*, pages 1204–1207, 2023.
3. Desirée Bill and Theodor Eriksson. Fine-tuning a llm using reinforcement learning from human feedback for a therapy chatbot application, 2023.
4. Yapei Chang, Kyle Lo, Tanya Goyal, and Mohit Iyyer. Boookscore: A systematic exploration of book-length summarization in the era of llms. *arXiv preprint arXiv:2310.00785*, 2023.
5. Samuel Kernan Freire, Chaofan Wang, and Evangelos Niforatos. Chatbots in knowledge-intensive contexts: Comparing intent and llm-based systems. *arXiv preprint arXiv:2402.04955*, 2024.
6. Jean-Christophe Golaz et al. The DOE E3SM model version 2: Overview of the physical model and initial model evaluation. *Journal of Advances in Modeling Earth Systems*, 14(12):e2022MS003156, 2022.
7. Shijia Huang, Jianqiao Zhao, Yanyang Li, and Liwei Wang. Learning preference model for llms via automatic preference data generation. In *The 2023 Conference on EMLP*, 2023.
8. Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. Evaluating llm-generated worked examples in an introductory programming course. In *Proceedings of the 26th Australasian Computing Education Conference*, pages 77–86, 2024.
9. Patrick Lewis, Perez, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.
10. Keith W Oleson, David M Lawrence, et al. Technical description of version 4.0 of the community land model (clm). *NCAR Tech. Note NCAR/TN-478+ STR*, 257:1–257, 2010.
11. Peter Schwartz, Dali Wang, Fengming Yuan, and Peter Thornton. SPEL: Software tool for porting e3sm land model with openacc in a function unit test framework. In *2022 Workshop on Accelerator Programming Using Directives (WACCPD)*, pages 43–51. IEEE, 2022.
12. Christos Tsigkanos, Pooja Rani, Sebastian Müller, and Timo Kehrer. Variable discovery with large language models for metamorphic testing of scientific software. In *ICCS*, pages 321–335, 2023.
13. Dave Van Veen, Cara Van Uden, et al. Clinical text summarization: Adapting large language models can outperform human experts. *Research Square*, 2023.
14. Tianyi Zhang, Faisal Ladhak, et al. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57, 2024.
15. Lianmin Zheng, Wei-Lin Chiang, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.
16. Weijian Zheng, Dali Wang, and Fengguang Song. XScan: an integrated tool for understanding open source community-based scientific code. In *ICCS*, pages 226–237. Springer, 2019.
17. Weijian Zheng, Dali Wang, and Fengguang Song. FQL: An extensible feature query language and toolkit on searching software characteristics for hpc applications. In *Tools and Techniques for High Performance Computing*, pages 129–142, 2020.