LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-849438

LM4HPC: Towards Effective Language Model Application in High-Performance Computing

L. Chen, P. Lin, T. Vanderbruggen, C. Liao, M. Emani, B. Supinski

May 22, 2023

19th International Workshop on OpenMP
Bristol, United Kingdom
September 13, 2023 through September 15, 2023

Disclaimer

LM4HPC: Towards Effective Language Model Application in High-Performance Computing

Le Chen[1,2], Pei-Hung Lin[1], Tristan Vanderbruggen[1], Chunhua Liao[1], Murali Emani[3], and Bronis de Supinski[1]

[1] Lawrence Livermore National Laboratory, Livermore CA 94550, USA
[2] Iowa State University, Ames, IA 50010, USA
[3] Argonne National Laboratory, Lemont, IL 60439, USA

Abstract. In recent years, language models (LMs), such as GPT-4, have been widely used in multiple domains, including natural language processing, visualization, and so on. However, applying them for analyzing and optimizing high-performance computing (HPC) software is still challenging due to the lack of HPC-specific support. In this paper, we design the LM4HPC framework to facilitate the research and development of HPC software analyses and optimizations using LMs. Tailored for supporting HPC datasets, AI models, and pipelines, our framework is built on top of a range of components from different levels of the machine learning software stack with Hugging Face-compatible APIs. Using three representative tasks, we evaluated the prototype of our framework. The results show that LM4HPC can help users quickly evaluate a set of state-of-the-art models and generate insightful leaderboards.

Keywords: Language model · Programming language processing · High-performance computing

1 Introduction

Language models (LMs) are models designed to understand and generate human language. In recent years, large language models (LLMs) trained on large amounts of text data have demonstrated stunning capabilities in various natural language processing and visualization tasks. They have also been widely used to process programming languages due to the similarities between natural languages and programming languages. For example, GPT-4 [5] shows early signs of artificial general intelligence. GitHub provides an AI assistant for developing software based on a large language model trained on code [10].

Given the rise of LLMs, it is natural for researchers and developers in the high-performance computing community to start exploiting LMs for addressing various challenges in HPC, including code analysis, code generation, performance optimization, question answering, and so on. However, mainstream frameworks of LMs were originally designed to serve natural language processing. It is difficult for newcomers in HPC to quickly access HPC-specific datasets, models, and pipelines. For example, the current popular Hugging Face platform does not

include dedicated pipelines for software analyses and optimizations. Another challenge is the entire field is evolving quickly, with new techniques emerging almost weekly, making it challenging for HPC users to keep up with the latest techniques and find relevant ones. Last, but not least, there is a lack of standard, reproducible evaluation processes for LMs focusing on HPC-specific tasks. Therefore, it is difficult to have a fair comparison among different models for a given HPC task.

This paper proposes a framework (named LM4HPC) designed to serve HPC users as first-class citizens by including internal components and external APIs relevant to HPC-specific tasks. LM4HPC's components include models, datasets, pipelines, and so on, while the APIs allow users to interact with these components to finish given HPC tasks. We highlight the contributions of our work as follows:

- We design an extensible framework for including and exposing relevant machine learning components to facilitate the adoption of large language models for HPC-specific tasks.
- The framework provides a set of APIs to facilitate essential operations, including code preprocessing, tokenization, integration with new data, and evaluation.
- A set of pipelines have been developed to support common HPC tasks, including code similarity analysis, parallelism detection, question answering, and so on.
- We provide HPC-specific datasets such as DRB-ML, OMP4Par, and OMPQA to support various HPC pipelines.
- Our work introduces standardized workflows and metrics to enable fair and reproducible evaluation of LLMs for HPC-specific tasks.
- Using three representative tasks, we demonstrated how the framework can be used to test a set of language models and generate leaderboards.

2 Background

Language models (LMs) are machine learning models designed to comprehend and generate human language. They can be used to facilitate natural and intuitive interactions between humans and machines. Early generations of LMs, using recurrent neural networks (RNNs), showed inspiring results for various natural language processing (NLP) tasks. A transformative evolution by the Transformer [24] reveals remarkable potentials of LMs. Introduced by Vaswani *et al.*, transformer models utilize the attention mechanism to capture the dependencies between all words in an input sentence, irrespective of their positions. Compared to RNNs, transformers process data in parallel rather than sequentially and significantly improve the efficiency of model training and inference. Transformers further enables the inauguration of large language models (LLMs). Compared to LMs, LLMs are trained on a vast amount of data and possess parameter counts on the order of billions or more, allowing them to generate more detailed and nuanced responses. Examples of LLMs include OpenAI's GPT-3, GPT-4, and Google's BARD. Nowadays, LLMs have shown remarkable capabilities in NLP tasks like translation, question answering, and text generation.

Table 1. Language models, associated training data and tasks

Model			Training data		Token	Avail.
Name	**Released**	**Size**	**Type**	**Size**	**Limit**	
BERT	2018/10	340M	Text	3.5B words	512	Weights
CodeBERT	2020/11	125M	Mixed	2.1M(bimodal) 6.4M (unimodal)	512	Weights
Megatron	2021/04	1T	Text	174 GB	512	Weights
GraphCodeBERT	2021/05	110M	Code	2.3M functions	512	Weights
CodeT5	2021/11	770M	Code	8.35M instances	512	Weights
GPT-3	2022/03/15	175B	Mixed	500B tokens	4096	Weights
LLaMA	2023/02/24	7∼65B	Mixed	1.4T tokens	4096	Weights*
GPT-4	2023/03/14	1T	Mixed	undisclosed	8k/32k	API*
BARD	2023/03/21	1.6B	Mixed	1.56T words	1000	API
Cerebras-GPT	2023/03/28	0.11∼13B	Text	800 GB	2048	Weights
Dolly 2.0	2023/04/12	3∼12B	Text	15k instr./resp.	2048	Weights
StarCoder	2023/05/4	15B	Code	1T tokens	8192	Weights
StarChat-Alpha	2023/05/4	16B	Code	31k instr./resp.	8192	Weights

Table 1 shows some example language models and their release dates, sizes, training data, input token length limits, and availability. LLaMA [23]'s weights can be obtained after filling out some form. GPT-4 has a waiting list to use its API. At the time of writing this paper, we have not yet obtained its access.

LMs are trained mainly by text data with a focus on NLP. The sources of the training data mainly come from books, web content, newspapers, scientific articles, and other text data in various natural languages. Latest LLMs have demonstrated rich skill sets in NLP including text prediction, common sense reasoning, reading comprehension, translation and question answering.

There has been a keen interest in deploying NLP techniques to programming language processing (PLP) tasks, such as code summarization, code generation, and code similarity analysis [8,14]. Previous studies have demonstrated successful applications of traditional language models to PLP tasks, showing the feasibility of this approach [12]. CodeBERT [13], for example, is a transformer-based model trained with a diverse range of programming languages and can be used for a variety of programming-related tasks. Similarly, CodeT5 [26] is a variant of Google's T5 language model, trained specifically on code datasets to perform advanced programming tasks like code completion, bug detection, and code summarization. Lately, StarCoder [18], a 15B parameter model trained with 1 trillion tokens sourced from a large collection of permissively licensed GitHub repositories, is developed to be a Large Language Model mainly for code generation or completion. StarChat-Alpha is a GPT-like chat model fine-tuned based on StarCoder to act as a helpful coding assistant.

2.1 LMs for HPC

With the recent breakthroughs in Generative Pretrained Transformer (GPT) large language models [4], it has become increasingly intriguing to explore the application of large language models (LLMs) for HPC tasks. However, their

deployment in the HPC domain is still relatively unexplored. This venture comes with various challenges, including:

1. Pipelines: Traditional language model frameworks like Hugging Face were designed to support natural language processing or compute vision problems. Expanding LMs to any new domain, including HPC, requires the addition of new pipelines designed to finish domain-specific tasks.
2. Datasets: The HPC domain encompasses an extensive amount of code spanning various fields, including biology and climate modeling. However, preparing this data for machine learning training, such as labeling parallelizable loops in HPC programs for parallelism detection, presents significant challenges. The scarcity of ready-to-use, pre-labeled HPC datasets poses a particular obstacle for training language models, especially large ones, highlighting the need for more shared resources in the community.
3. Pre-processing: Pre-processing in the context of LMs for HPC typically involves the conversion of source files into a sequence of tokens. However, the direct application of NLP tokenizers to code can be sub-optimal. For instance, an NLP tokenizer might split a variable name into two tokens, a scenario undesirable for PLP analysis. Also, models designed for processing source code may use graph representations, such as abstract syntax trees, to have better performance.
4. Input size limit: Language models often have limited input token lengths (such as 512 to a few thousand of tokens). HPC tasks often involve processing large-scale software packages with millions of lines of source code.
5. Evaluation: There is a pressing need for standardized and reproducible evaluation of different models in the context of various HPC tasks, using metrics suitable for domain-specific requirements.

3 Approach

To address the challenges discussed in Section 2, we introduce LM4HPC, a comprehensive framework that encapsulates a suite of machine learning components within user-friendly APIs. This framework is tailored for HPC users, simplifying the implementation process and making the robust capabilities of language models more accessible and user-friendly within the HPC community. The primary goal of LM4HPC is to reduce the complexities inherent in employing language models, thus enabling HPC users to leverage their powerful capabilities more effectively and efficiently.

3.1 LM4HPC design overview

Figure 1 provides the overview of the LM4HPC framework. It is built on top of multiple internal machine learning components with Hugging Face-compatible APIs. Higher-level components provide concepts and interfaces to users, while middle or lower-level components provide implementation support. Table 2 shows the example classes and functions in LM4HPC API, including those supporting

HPC-specific language models, tokenizers for programming languages, datasets, inference pipelines, and evaluation. We elaborate on some essential components in the following subsections.

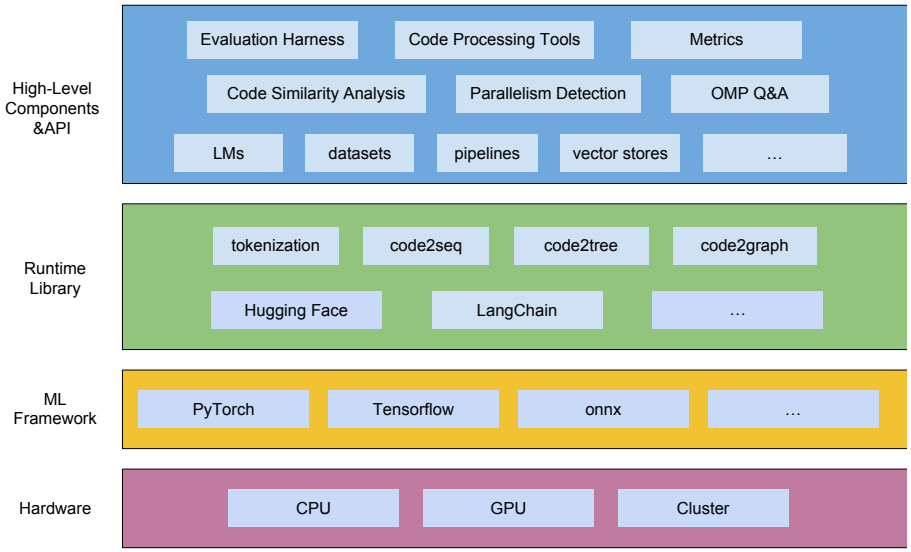

Fig. 1. Overview of the LM4HPC framework

3.2 HPC Tasks and Inference Pipelines

HPC users are interested in a wide range of tasks related to programming language processing. Table 3 outlines one way to categorize HPC-specific tasks. The purpose here is not to provide a comprehensive taxonomy of all tasks but a starting point for common tasks we are interested in supporting in our framework. Most tasks are self-explanatory by names and each may have further sub-tasks. For example, clone detection can be viewed as a specialized sub-task under code similarity analysis.

In the context of machine learning, a pipeline represents a sequence of data processing stages to complete a task. Our LM4HPC framework extends the pipeline function provided by Hugging Face, adapting it for HPC tasks. We have developed three inference pipelines: code similarity analysis, parallelism detection, and OpenMP question answering. Code similarity analysis determines the similarity between a pair of code snippets. Parallel detection is defined to check if an input code snippet can be parallelized or not using OpenMP. The OpenMP question answering pipeline is designed to use models to generate answers to OpenMP-related questions.

Table 2. LM4HPC API: example classes and functions. Each class can be imported using "from lm4hpc import *" in Python.

LM4HPC classes	**Description**	**Example API functions**
`hpcmodel`	Fine-tune text-based (HF, OpenAI) and graph-based models, including local private ones, for HPC tasks	`hpcmodel.from_pretrained(model_name_or_path: Optional[str], *model_args, **kwargs)`
		`hpcmodel.save_pretrained(model_name_or_path: str, *model_args, **kwargs)`
		`hpcmodel.finetune()`
`hpctokenizer`	APIs to represent code in either tokenized text, trees, or graphs	`hpctokenizer.from_pretrained(model_name_or_path: Optional[str], *model_args, **kwargs)`
		`hpctokenizer.addtokens(contentsingle_word=False, strip=False, normalized=True)`
		`hpctokenizer.encoding()`
`hpcdatasets`	Load and process HPC datasets	`hpcdatasets.load(path: str, data_files: Union[str, List, Dict, None], **kwargs)`
		`hpcdatasets.split(dataset: hpcdatasets, partition: [float, float, float], **kwargs)`
		`hpcdatasets.shuffle(dataset: hpcdatasets, **kwargs)`
		`hpcdatasets.sort(dataset: hpcdatasets, **kwargs)`
`hpcpipeline`	Pre-built pipelines for common PLP tasks	`hpcpipeline(task: str, model_name_or_path: str, *model_args, **kwargs)`
`hpceval`	Evaluate the results of various models	`hpceval.compute(task: str, models_name_or_path: [[str]], data_files: Union[str, List, Dict, None], *model_args, **kwargs)`
		`hpceval.plot(shape: str)`

Table 3. HPC Tasks for Programming Language Processing: Categories and Examples

Code Analysis	**Code Generation**	**Others**
Compiler Analysis	Code Completion	Test Case Generation
Algorithm Classification	Natural Language-to-Code	Code Search
Code Similarity Analysis	Code Translation	Question Answering
Documentation Generation	Code Repair	Code Review
Parallelism Detection	Code Migration	Decompilation
Defect Detection	Compilation	IR-to-Source Translation

Tokenizers are responsible for preprocessing input into an array of numbers as inputs to a model. They are essential components used by pipelines. Most LM tokenizers are primarily designed for NLP tasks. For instance, given a function name `my_func`, a typical NLP tokenizer like BERT might split it into separate tokens (such as '`my`', '_', and '`func`') while a code-aware tokenizer may treat the function name as a single entity to ensure a more meaningful representation.

To overcome this, we developed the LM4HPC tokenizer, leveraging the treesitter [2] and programl [1] library. Our tokenizer is specifically designed to handle the pre-processing of code data required for a language model. It includes tokenizers such as the ast-tokenizer. As a result, LM4HPC can accommodate models (such as augAST [9]) that require AST as input in the pipeline.

3.3 Datasets

Datasets are crucial for any machine learning application. Within the LM4HPC framework, we contribute HPC-specific datasets either by converting existing ones into Hugging Face-compatible formats or by creating new ones from scratch.

We have converted three existing datasets to be compatible with Hugging Face dataset API: POJ-104 [21], DRB-ML [20], and OMP4Par [9]. POJ-104 is derived from a pedagogical programming open judge (OJ) system that automatically evaluates the validity of submitted source code for specific problems by executing the code. This dataset is particularly useful for the code similarity task. The DRB-ML dataset contains 658 C/C++ OpenMP kernels derived from DataRaceBench [19]. We extended it to have labels indicating if a kernel is parallelizable or not. The OMP4Par dataset is an open-source benchmark composed of data from three resources: code crawled from GitHub, OpenMP benchmarks such as Nas Parallel Benchmarks [17] and Rodinia [7], and synthetic code. This dataset contains loops with labels indicating whether a loop is parallel and, if parallelizable, the corresponding OpenMP directive associated with the loop.

Furthermore, we have manually created a new OpenMP question answering dataset called OMPQA in order to probe the capabilities of language models in single-turn interactions with users. Similar to other QA datasets, we include some request-response pairs which are not strictly question-answering pairs. The categories and examples of questions in the OMPQA dataset can be found in Table 4.

Table 4. OMPQA: categories and examples of questions

Category	Count	Example Questions
Basics	40	What is a worksharing construct in OpenMP?
Examples	20	Give an example OpenMP C code for computing PI using numerical integration.
Compilers	24	In what language is LLVM written? How is a parallel region represented in Clang?
Benchmarks	23	What are the NAS Parallel benchmarks? Which benchmark assesses data race detection tools?

3.4 Integration With New Data

Language models derive knowledge from training datasets and store this knowledge in internal weights within the model's neural network architecture. However, incorporating new information into a trained model presents a challenge. Traditionally, one might fine-tune pre-trained models with their own data for specific tasks, but this approach requires substantial relevant data and can be resource-intensive. An alternative approach involves integrating new data as context information into a user prompt, but this is constrained by the limited input token lengths of current models.

To address this challenge, LM4HPC leverages the LangChain framework [6] to easily integrate new data. LangChain aggregates a wide variety of components

to build applications using LLMs. Particularly, it provides APIs allowing LLM applications to store large amounts of text in semantic databases called vector stores. The way to integrate new data can be done in two steps. First, text data is chunked and embedded with an LLM before being saved into a vector store. Later, user prompts are matched with relevant chunks in the vector store using similarity analysis. The top-matched chunks are then injected into the original prompts to form a new prompt with relevant context information. By employing this new prompt, language models can generate answers that incorporate new and relevant user data while still staying within the token length limits.

3.5 Evaluation

An easily accessible harness for evaluating different language models on HPC tasks is crucial. Standard and reproducible results from such evaluations can provide researchers and developers with insightful starting points, helping them select suitable models for their specific needs and identify research opportunities.

In response to this need, we developed an evaluator API in LM4HPC. One challenge we encountered is the lack of standardized metrics for code evaluation. Unlike natural language tasks, where metrics such as BLEU, ROUGE, and METEOR are commonly used, the code domain lacks such universally accepted quality measures. We are adding various LLM metrics such as CodeBLEU [22] for code output. Another challenge is that language models may generate different answers for the same input in different inference runs. Evaluation should consider consistent sampling settings (such as temperatures) and control over random seeds to improve reproducibility.

Ultimately, many users are interested in seeing leaderboards that showcase mainstream models competing on common HPC tasks. To satisfy this interest, we create and release a set of test harnesses scripts to enable standard and reproducible evaluation for supported HPC tasks.

4 Preliminary Results

In this section, we evaluate the current prototype implementation of LM4HPC through experiments designed to generate leaderboards for three representative tasks: Code Similarity Analysis, Parallelism Detection, and OpenMP Question Answering. LM4HPC utilizes LangChain v0.0.174, Hugging Face's transformers v4.29.0 and datasets v2.12.0 as our runtime libraries. Details of the models and datasets will be discussed in subsequent subsections.

Our experiments were conducted on two machines: 1) a Google Colab VM with a 6-core Xeon processor operating at 2.20GHz, 83.5 GB main memory, 166GB HDD drive, and an NVIDIA A100 GPU with 40 GB memory. 2) a Dell workstation equipped with a dual Intel Xeon 6238 CPU operating at 2.10GHz, 128 GB main memory, 1TB SSD drive, and an NVIDIA Quadro RTX 6000 GPU with 24GB memory. The majority of our experiments were run on the Google Colab machine to leverage its superior GPU memory. However, we encountered difficulties running Cerebras-GPT on the Colab machine and were compelled to use the Dell workstation with larger CPU memory instead.

4.1 Code Similarity Analysis

The code similarity task is designed to measure the syntactic and/or semantic similarity between two code snippets. Such analysis information can be beneficial in various scenarios such as plagiarism detection, code reuse and refactoring, bug detection and repair, licensing compliance, malware detection, and so on.

Preparing Datasets and Ground Truth. Two datasets introduced in Section 3.3, POJ-104 and DRB-ML, are loaded through LM4HPC's datasets API for this experiment. For each pair of code snippets in the POJ-104 dataset, we assign a binary similarity label based on their functional labels. A similarity label of 1 signifies that the snippet pair shares the same functional label and we assign a similarity score of 1. Otherwise, the label is 0. We have processed the DRB-ML dataset using a similar methodology to generate code pairs and labels. The main difference is that the similarity ground truth for DRB-ML is derived from its own similarity score table [11], providing a precise and reliable similarity measurement between code snippets in the dataset.

Inference Experiments and Evaluation. We employ LM4HPC's code similarity pipeline to test various models. The default model for this pipeline is CodeBERT. We additionally select four models from Table 1 for evaluation: GraphCodeBERT, gpt-3.5-turbo, Dolly 2.0 (12B), and Cerebras-GPT (13B). We set the maximum token length for the model output to 256. This limits the verbosity of the model and keeps its responses concise. Additionally, we set the temperature parameter to 0 when applicable. For models like Dolly 2.0 that require positive temperature, we set the temperature to be 1×10^{-6}. This setting ensures that the model's responses are consistent and deterministic, minimizing variability and uncertainty in its output.

Within LM4HPC, the approach of processing input code pairs depends on the type of the model employed. Models like CodeBert and GraphCodeBert are specifically devised and trained on a variety of programming languages. We directly feed a pair of code snippets to generate a similarity prediction. On the other hand, large language models like gpt-3.5-turbo, Dolly 2.0, and Cerebras-GPT are evaluated using the following prompt template: "Code 1: {...} Code 2: {...} Determine whether the two code snippets are similar. If the code snippets are similar, output 1; otherwise, output 0.".

Results. The Code Similarity Analysis leaderboards generated using the two datasets are shown in Table 5. Notably, gpt-3.5-turbo demonstrates superior performance. Two other models, StarChat-Alpha and Dolly 2.0, also exhibit commendable performance. Most large language models outperform traditional models (GraphCodeBERT and CodeBERT) that were specifically trained for code analysis. However, Cerebras-GPT struggled to comprehend the code and mostly returned arbitrary word tokens, indicating a lack of effective code understanding since it is mostly designed for natural language processing.

Table 5. Code Similarity Analysis Leaderboard: POJ-104 (left) and DRB-ML (right)

Model	Precision	Recall	F1	Model	Precision	Recall	F1
gpt-3.5-turbo	**78.4**	**74.2**	**76.2**	**gpt-3.5-turbo**	**82.4**	**81.3**	**81.8**
Dolly 2.0 12B	61.9	61.3	61.6	StarChat-Alpha	79.6	77.4	78.5
StarChat-Alpha	59.4	56.2	57.8	Dolly 2.0 12B	74.3	73.2	73.7
GraphCodeBERT	52.7	60.3	56.3	GraphCodeBERT	79.4	77.9	78.6
CodeBERT	51.5	59.4	55.2	CodeBERT	76.9	74.5	75.7
Cerebras-GPT 13B	0	0	0	Cerebras-GPT 13B	0	0	0

4.2 Parallelism Detection

The parallelism detection task aims to identify parallelism opportunities within a given code snippet. We utilized two datasets, OMP4Par and DRB-ML introduced in Section 3.3, for the experiment.

Preparing Datasets and Ground Truth. The OMP4Par dataset is specifically designed for parallelism detection. Its existing labeling scheme allows us to prepare the data for binary classification models. Similarly, we prepared DRB-ML dataset with a label indicating whether each code snippet is parallelizable using OpenMP or not.

It is worth noting that both datasets have undergone source code pre-processing steps, including comment removal and code snippet extraction. These steps are common practice [9] to ensure that code snippets are small enough to be fed into language models with limited input token sequence sizes. However, the resulting code snippets may lose their context information, such as variable declarations. This is a serious limitation of language models with limited input sizes when applied to process large source files.

Inference Experiments and Results. We selected six models to generate parallelism detection leaderboards. Four of them are introduced in Section 2. They take the code snippets in a prompt template: "As an OpenMP expert, you will analyze the given code snippet to determine if it can be parallelized. Code: {...}. Answer yes or no first:". The other two are augAST [9] and DeepSCC-based [15], which are pre-trained models using OMP4Par's training dataset. We fed code snippets to these two models to directly obtain predicted labels.

Table 6 presents the resulting leaderboards. The highest F1 score reaches 93.9, indicating that LMs can be very effective for detecting parallelism. However, the datasets contain small-scale code snippets that are significantly simpler than real HPC codes. Again, gpt-3.5-turbo outperforms all other models overall, including specially trained models like augAST and DeepSCC. AugAST performs better than gpt-3.5-turbo in terms of precision, suggesting it's more effective in predicting a positive class, which, in this case, is parallelizable code. Finally, Cerebras-GPT did not perform well in this code analysis task.

4.3 OpenMP Q&A

In this experiment, we utilized LM4HPC to evaluate the capabilities of several language models in answering questions related to OpenMP. This evaluation was conducted using the OMPQA dataset, introduced in Section 3.3.

Table 6. Parallelism Detection Leaderboards: OMP4Par (left) and DRB-ML(right)

Model	Precision	Recall	F1	Model	Precision	Recall	F1
gpt-3.5-turbo	90.6	**89.3**	**89.9**	**gpt-3.5-turbo**	90.0	**98.9**	**94.2**
augAST	**92.1**	82.4	87.0	augAST	**91.4**	72.3	80.7
DeepSCC	82.7	81.4	82.0	DeepSCC	80.4	79.5	79.9
StarChat-Alpha	85.7	68.2	75.9	StarChat-Alpha	81.9	20.3	32.5
Dolly 2.0 12B	64.2	63.7	63.9	Dolly 2.0 12B	40.0	11.2	2.17
Cerebras-GPT 13B	0	0	0	Cerebras-GPT 13B	0	0	0

Experiment Settings. Each model receives the question in the following prompt template: "You are an OpenMP expert. Please answer this question. Question: {question}". Two widely adopted metrics are selected to evaluate the quality of answers: the Bilingual Evaluation Understudy (BLEU) and ROUGE-L metrics. BLEU is a precision-oriented metric measuring the overlap of n-grams between the generated text and a set of reference texts. ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation - Longest Common Subsequence) calculates the longest common subsequence (LCS) that appears in a left-to-right sequence in both the system-generated and reference summaries, thus providing a measure of the coherence and fluidity of the generated text. We utilized the averaged ROUGE-L scores to assess the sentences across the generated output.

Results. Table 7 displays the Q&A leaderboard of several selected models. We additionally include the memory and execution time information. gpt-3.5-turbo does not consume any local GPU memory as it is invoked remotely through OpenAI's API.

Again, gpt-3.5-turbo unsurprisingly outperforms other LLMs. However, the highest average ROUGE-L F1 score of 0.259 indicates that all models have room for improvement in answering OpenMP questions. One reason is that many questions in OMPQA are open-ended and do not necessarily have a single correct answer. Also, the two metrics used do not sufficiently consider semantics.

Table 7. Q&A Leaderboard using the OMPQA dataset. The arrow indicates the performance changes when augmenting the external knowledge base by LangChain.

Model	CPU Mem. (GB)	GPU Mem. (GB)	Time(s)	BLEU	ROUGE-L(AVG)		
					Recall	Precision	F1
gpt-3.5-turbo + LangChain	4.1	0	21.452	**0.147**↑	0.347↓	0.262↑	**0.259**↑
gpt-3.5-turbo	4.2	0	12.749	0.139	**0.446**	0.231	0.257
StarChat-Alpha	6.8	18.9	29.732	0.082	0.322	0.149	0.172
Dolly 2.0 12B + LangChain	27.4	39.8	7.217	0.084↑	0.228↑	0.232↓	0.182↑
Dolly 2.0 12B	27.1	39.2	8.147	0.06	0.208	**0.312**	0.148
Cerebras-GPT 13B	52.6	11.7	590.763	0.071	0.319	0.089	0.112

To enhance the capacity of large language models (LLMs) in accurately responding to OpenMP queries, we integrate the official OpenMP documentation

into our process. We employ LangChain, a mechanism designed to efficiently store and retrieve language model embeddings, enabling us to accommodate large volumes of new data. To assess the efficacy of using LangChain to incorporate additional user data, we leverage its API to create a vector store. This vector store holds embeddings of text chunks derived from the OpenMP API Specification v5.2 (669 pages) and the OpenMP Application Programming Interface Examples v5.2.1 (575 pages). We then select two LangChain-supported models, GPT-3.5 and Dolly 2.0, to utilize the vector store as an additional resource for answering queries, thereby demonstrating the practical utility of our approach. The results indicate slight improvements in the BLEU and ROUGE-L F1 scores, increasing from 0.139 to 0.147 and 0.257 to 0.259, respectively. However, there are mixed results for recall and precision metrics. GPT-3.5-turbo demonstrates a higher recall of 0.446, surpassing the 0.347 achieved by the LangChain approach. Dolly 2.0 has better precision, 0.312, as opposed to the LangChain approach's 0.232.

Further, we examine the effectiveness of the LangChain approach across different question categories. When addressing 'Basic' questions, the BLEU scores rise by 20.7% and 9.8% for gpt-3.5-turbo and Dolly 2.0, respectively. Additionally, we assess the LangChain performance using the CodeBLEU metric[22] for the 'Examples' category, observing a score increase of 6.1% and 12.2% for gpt-3.5-turbo and Dolly 2.0, respectively. These observations indicate that augmenting LLMs with documentation via LangChain improves performance for the 'Basic' and 'Examples' categories. However, for the 'Compilers' and 'Benchmarks' categories, gpt-3.5-turbo and Dolly 2.0's performance diminishes when using LangChain, recording an average BLEU score drop of 8.0% and 7.9%, respectively. This drop is likely because our documentation does not include information relevant to compiler and benchmark topics.

We also manually investigated the answers generated by the models. Overall, StarChat-Alpha delivers competitive results compared to GPT-3.5. It seems to be a good choice for people who want to use open-source language models based on our experiments. Research has indicated that GPT-4 surpasses GPT-3.5 in a variety of domains. However, as of now, API accessibility for GPT-4 has not been made publicly available. We plan to assess GPT-4's performance as soon as it becomes accessible and incorporate it into our framework if it benefits HPC tasks.

5 Related Work

PyTorch and TensorFlow are the most popular frameworks backed by Meta AI and Google, respectively. Both frameworks are similar in many respects, including 1) providing low-level APIs for development, 2) supporting a rich collection of libraries, and 3) maintaining dedicated hubs - PyTorch Hub and TensorFlow Hub - for providing pre-trained ML models. Hugging Face is a large open-source community that builds tools to enable users to build, train, and deploy machine learning models based on open-source code and technologies. Hugging Face is

best known for its Transformers library, which exposes a collection of Python APIs to leverage state-of-the-art deep learning architectures for NLP tasks. To simplify end-to-end NLP tasks, Hugging Face Transformers offers a pipeline that performs all pre- and post-processing steps on the given input text data. The overall process of the model inference is encapsulated within these pipelines. With the pipeline, users only need to provide the input texts and the model for the task. The remaining connections among a model and required pre- and post-processing steps are hidden within the pipeline implementation.

Various research works and developments have been conducted to improve the ML ecosystem to be Findable, Accessible, Interoperable, and Reproducible (FAIR). These existing frameworks aim to make the models, datasets, or both FAIR. Among these frameworks, HPCFAIR [25] focuses on supporting model interoperability, search capabilities for datasets and models, and seamless integration into HPC workflows. The work in [28] extended this work to include support for interoperability across different framework implementations using ONNX and provision to retrain a model with transfer learning. However, the HPCFAIR framework relies on users to handle data pre- and post-processing. In comparison, LM4HPC is equipped to manage data processing within the pipeline design and generate leaderboards for supported HPC tasks.

General LLMs are trained with data covering general knowledge and information that is usually collected from public domains. Domain-specific datasets can be collected for the training of a specialized model or the fine-tuning of a general-purpose model. MedQA[16] is an example of domain-specific datasets collecting question-answer pairs and textbooks from professional medical board exams. ExeBench [3], another domain-specific dataset for compilation and software engineering tasks, contains millions of runnable and representative C functions collected from GitHub. In addition to collecting existing data, ML research has started automating dataset creation with LLMs' assistance. The developers of LaMini-LM [27] develop a large set of 2.58M instruction and response pairs based on both existing and newly-generated instructions. A handful of seed examples from the existing LLM prompts and 2.2M categories from Wikipedia from existing are submitted to the gpt-3.5-turbo model to generate relevant instructions. Similarly, the responses for the generated instructions are also generated by gpt-3.5-turbo.

6 Conclusion

In this paper, we presented our efforts to facilitate the application of language models for tasks specific to High-Performance Computing. We have developed the LM4HPC framework to encompass and expose relevant machine learning components via corresponding APIs. Our experimental findings suggest that GPT-3 performs competitively, despite not being specifically designed for HPC tasks. However, there is significant room for improvement in answering OpenMP questions. Furthermore, the input size limitation of language models adds complexity to certain tasks, such as parallelism detection. Finally, an obstacle to

advancing the application of language models for HPC tasks is the absence of HPC-specific training and evaluation datasets.

Our future work will explore automated approaches to generating HPC-specific datasets. We intend to enhance LM4HPC's capabilities to support the fine-tuning of models for HPC-related tasks, including those related to the Message Passing Interface (MPI), and to provide performance analysis and optimization suggestions.

Acknowledgement

Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-CONF-849438) and supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research. This research was also funded in part by and used resources at the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

1. ProGraML: Program Graphs for Machine Learning (2022), https://pypi.org/project/programl/, accessed: 2023-05-15
2. The py-tree-sitter project (2023), https://pypi.org/project/tree-sitter-builds/, accessed: 2023-05-15
3. Armengol-Estapé, J., Woodruff, J., Brauckmann, A., Magalhães, J.W.d.S., O'Boyle, M.F.: ExeBench: an ML-scale dataset of executable C functions. In: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming. pp. 50–59 (2022)
4. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
5. Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S., et al.: Sparks of artificial general intelligence: Early experiments with gpt-4. arXiv preprint arXiv:2303.12712 (2023)
6. Chase, H.: LangChain: Next Generation Language Processing (2023), https://langchain.com/, accessed: 2023-05-15
7. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H., Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing. In: 2009 IEEE international symposium on workload characterization (IISWC). pp. 44–54. Ieee (2009)
8. Chen, L., Mahmud, Q.I., Jannesari, A.: Multi-view learning for parallelism discovery of sequential programs. In: 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 295–303. IEEE (2022)
9. Chen, L., Mahmud, Q.I., Phan, H., Ahmed, N.K., Jannesari, A.: Learning to Parallelize with OpenMP by Augmented Heterogeneous AST Representation. arXiv preprint arXiv:2305.05779 (2023)
10. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)

11. Chen, W., Vanderbruggen, T., Lin, P.H., Liao, C., Emani, M.: Early Experience with Transformer-Based Similarity Analysis for DataRaceBench. In: 2022 IEEE/ACM Sixth International Workshop on Software Correctness for HPC Applications (Correctness). pp. 45–53. IEEE (2022)
12. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
13. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al.: Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155 (2020)
14. Flynn, P., Vanderbruggen, T., Liao, C., Lin, P.H., Emani, M., Shen, X.: Finding Reusable Machine Learning Components to Build Programming Language Processing Pipelines. arXiv preprint arXiv:2208.05596 (2022)
15. Harel, R., Pinter, Y., Oren, G.: Learning to parallelize in a shared-memory environment with transformers. In: Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming. pp. 450–452 (2023)
16. Jin, D., Pan, E., Oufattole, N., Weng, W.H., Fang, H., Szolovits, P.: What Disease does this Patient Have? A Large-scale Open Domain Question Answering Dataset from Medical Exams. arXiv preprint arXiv:2009.13081 (2020)
17. Jin, H.Q., Frumkin, M., Yan, J.: The OpenMP implementation of NAS parallel benchmarks and its performance (1999)
18. Li, R., Allal, L.B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., et al.: Starcoder: may the source be with you! arXiv preprint arXiv:2305.06161 (2023)
19. Liao, C., Lin, P.H., Asplund, J., Schordan, M., Karlin, I.: Dataracebench: a benchmark suite for systematic evaluation of data race detection tools. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–14 (2017)
20. Lin, P.H., Liao, C.: Drb-ml-dataset (12 2022). https://doi.org/10.11579/1958879
21. Mou, L., Li, G., Zhang, L., Wang, T., Jin, Z.: Convolutional neural networks over tree structures for programming language processing. In: Proceedings of the AAAI conference on artificial intelligence. vol. 30 (2016)
22. Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A., Ma, S.: CodeBLEU: a method for automatic evaluation of code synthesis. arXiv preprint arXiv:2009.10297 (2020)
23. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al.: Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971 (2023)
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
25. Verma, G., Emani, M., Liao, C., Lin, P.H., Vanderbruggen, T., Shen, X., Chapman, B.: HPCFAIR: Enabling FAIR AI for HPC Applications. In: 2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC). pp. 58–68. IEEE (2021)
26. Wang, Y., Wang, W., Joty, S., Hoi, S.C.: Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. arXiv preprint arXiv:2109.00859 (2021)

27. Wu, M., Waheed, A., Zhang, C., Abdul-Mageed, M., Aji, A.F.: LaMini-LM: A Diverse Herd of Distilled Models from Large-Scale Instructions. arXiv preprint arXiv:2304.14402 (2023)
28. Yu, S., Emani, M., Liao, C., Lin, P.H., Vanderbruggen, T., Shen, X., Jannesari, A.: Towards Seamless Management of AI Models in High-Performance Computing (2022)