LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-853160

Data Race Detection Using Large Language Models

L. Chen, X. Ding, P. Lin, C. Liao

August 14, 2023

Correctness 2023: Seventh International Workshop on Software Correctness for HPC Applications
Denver, CO, United States
November 12, 2023 through November 12, 2023

Disclaimer

Data Race Detection Using Large Language Models

Le Chen
lechen@iastate.edu
Iowa State University
Ames, IA, USA
Lawrence Livermore National Laboratory
Livermore, CA, USA

Xianzhong Ding
xding5@ucmerced.edu
University of California, Merced
Merced, CA, USA
Argonne National Laboratory
Lemont, IL, USA

Murali Emani
memani@anl.gov
Argonne National Laboratory
Lemont, IL, USA

Tristan Vanderbruggen,
Pei-Hung Lin, Chunhua Liao
{vanderbrugge1,lin32,liao6}@llnl.gov
Lawrence Livermore National Laboratory
Livermore, CA, USA

ABSTRACT

Large language models (LLMs) are demonstrating significant promise as an alternate strategy to facilitate analyses and optimizations of high-performance computing programs, circumventing the need for resource-intensive manual tool creation. In this paper, we explore a novel LLM-based data race detection approach combining prompting engineering and fine-tuning techniques. We create a dedicated dataset named DRB-ML, which is derived from DataRaceBench, with fine-grain labels showing the presence of data race pairs and their associated variables, line numbers, and read/write information. DRB-ML is then used to evaluate representative LLMs and fine-tune open-source ones. Our experiment shows that LLMs can be a viable approach to data race detection. However, they still cannot compete with traditional data race detection tools when we need detailed information about variable pairs causing data races.

KEYWORDS

data race detection, large language model, OpenMP

ACM Reference Format:
Le Chen, Xianzhong Ding, Murali Emani, and Tristan Vanderbruggen,, Pei-Hung Lin, Chunhua Liao. 2023. Data Race Detection Using Large Language Models. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3624062.3624088

1 INTRODUCTION

The advent of many-core and GPU-accelerated systems has fostered the need for threaded programming models, such as OpenMP and CUDA, to exploit intra-node parallelism effectively. However, a perennial challenge accompanying these programming models is the risk of data races. Data races transpire when two or more threads access the same memory location simultaneously in a conflicting manner, without sufficient synchronization, with at least one of these accesses involving a write operation. This type of bug induces unpredictable behaviors in code, meaning they may not consistently manifest each time the code is executed, thereby exacerbating their detection and resolution difficulties.

Various tools, such as Intel Inspector [15] and ThreadSanitizer [22], have been developed to assist developers in responding to the challenges associated with data race detection in multithreaded programs. Due to the fast evolution of parallel programming, these tools need to be constantly re-evaluated and manually updated to support new language features and code patterns. A dataset explicitly designed for data race analysis, named DataRaceBench (DRB) [21], was introduced for evaluating the performance and effectiveness of various detection tools and methodologies.

In recent years, the realm of machine learning has been illuminated by significant breakthroughs, particularly the emergence of Large Language Models (LLMs). Harnessing the power of deep learning [11], LLMs have demonstrated their proficiency in comprehending and generating human-like text from provided prompts. Given the remarkable ability of LLMs to understand and generate text, they hold immense potential in the field of Programming Language Processing (PLP) tasks [5, 6, 18, 30]. This includes but is not limited to tasks such as code analysis, generation, and bug detection, showcasing an exciting expansion beyond the traditional confines of Natural Language Processing (NLP).

Building on their numerous applications in PLP, LLMs present potential for deployment in the specialized area of data race detection. We envisage that a fine-tuned LLM for data race detection could discern common patterns and contexts that precipitate data races. When applied to unfamiliar code, it could forecast potential data race conditions, thus equipping developers with a proactive alert mechanism to mitigate possible risks. Compared to traditional static or dynamic analysis methods, if adequately trained, machine learning approaches impose minimal human labor and runtime overhead and can effectively respond to a broad spectrum of data race patterns. Furthermore, with their innate capability to generate human-like text, LLMs could facilitate detailed explanations about detected data race conditions. Such insights could guide developers in discerning the underlying causes of these bugs and subsequently aid in devising more effective resolution strategies.

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3624088

The application of Large Language Models in data race detection represents a pioneering field of research. Our work commences by generating a comprehensive dataset distinctly labeled with explicit data dependencies and data race information, emphasizing our commitment to ensuring the accuracy and dependability of the model's output. Subsequently, we propose novel experimental approaches for data race detection using LLMs. The following contributions distinguish this paper:

(1) We have derived an innovative dataset from DataRaceBench for machine learning training and large language model fine-tuning.
(2) We extensively evaluate several prominent LLMs using various prompting techniques.
(3) We fine-tune LLMs for the explicit task of data race detection, thereby enhancing their predictive accuracy.
(4) A detailed comparative study between traditional data race detection tools and LLM-based methods, highlighting their strengths and weaknesses.

2 BACKGROUND

In this section, we provide an overview of large language models and their application in the context of programming language processing. We also introduce DataRaceBench.

2.1 Large Language Models

Large Language Models (LLMs) are large-sized machine learning models specifically designed to perform various natural language processing tasks, such as analyzing and generating text, answering questions in a conversational manner, and translating text from one language to another. Previous work [33] observed that large-sized pre-trained language models exhibit behaviors distinct from smaller ones (e.g., 330M-parameter BERT and 1.5B-parameter GPT-2) and show surprising abilities (called emergent abilities). Large language models have emerged as revolutionary tools in machine learning. The surging popularity of LLMs can be attributed to their versatile applicability and unparalleled performance in diverse tasks. LLMs have consistently outperformed traditional models, from enhancing natural language processing applications like sentiment analysis [31] and chatbots [17] to aiding researchers in content generation, summarization, and translation [16, 19].

Despite their inherent proficiency in context-dependent learning, pre-trained LLMs often require additional training or fine-tuning to perform specialized or novel tasks. This process allows the models to adapt to specific problem domains, thereby improving their performance and relevance in a given context.

Integrating Natural Language Processing (NLP) techniques in Programming Language Processing (PLP) tasks has sparked substantial interest. With applications that extend to code summarization, code generation, and code similarity analysis [4, 14], this emerging field has witnessed the successful deployment of traditional language models, underscoring the viability and potential of this approach [8].

A remarkable advancement in this domain is the adaptation of transformer-based language models for PLP tasks. Representative models, such as CodeBERT [13] and CodeT5 [29], epitomize this trend. These models leverage a transformer architecture and are trained on a wide array of programming languages to facilitate an extensive spectrum of programming-related tasks.

In the context of Large Language Models for Code (Code LLMs), several works [3, 9, 25] have explored pre-trained LLMs, either general-purpose or task-specific, for PLP tasks. In this study, we focus on four representative LLMs: GPT-3.5-turbo, GPT-4, Llama2-7b, and StarChat-beta (StarChat-β), which demonstrate diverse capabilities and applications in the sphere of code analysis and generation.

GPT-3.5-turbo [2], engineered by OpenAI, is a state-of-the-art language model capable of generating human-like text and comprehending nuanced prompts. For our research, we employ the 16k version of the model, accommodating up to 16384 input tokens. Succeeding GPT-3.5-turbo, GPT-4 [23] marks OpenAI's latest and most powerful offering. While GPT-4 retains the fundamental architecture of its predecessor, it capitalizes on expanded training across a diverse range of internet text, thereby enhancing both the model's size and capabilities. Llama2 [27] is one of the latest models released by Meta. As a substantial and robust language model, it demonstrates particular strength in tasks requiring deep understanding and information synthesis. Based on StarCoder [20], StarChat constitutes a series of GPT-style models explicitly crafted for code-related tasks. Their base models are 15.5B parameter models trained on 80+ programming languages. StarChat-beta is the second model in this series with 16B parameters.

An LLM-based approach offers distinct advantages, including the capacity to automatically capture common patterns across similar languages and to avoid the need for manual tool development for individual languages. Compared with previous machine learning-based approaches [26], LLMs can be continually fine-tuned on new data, adapting to new domains or specific tasks while still retaining their broad capabilities. However, the potential of LLMs in the realm of data race detection has yet to be fully explored. This research aims to probe into and unravel their capabilities in this domain.

2.2 Data Race Detection and DataRaceBench

Prominent techniques for data race detection leverage two major techniques: static and dynamic analysis. Static analysis tools such as Locksmith [24], RELAY [28], and ompVerify [1] inspect program source code or intermediate representations (IRs) to reveal potential data races through control flow and data dependency analysis. On the other hand, dynamic analysis tools such as Inter Inspector [15] and ThreadSanitizer [22] inspect program behavior during execution by instrumenting code to monitor memory accesses in real time. This is achieved by instrumenting the code to observe memory access in real time. Techniques under dynamic analysis, such as lockset-based and happens-before-based detection, often yield better accuracy in data race detection.

Despite the accurate output, dynamic analysis methods introduce runtime overhead and will likely miss certain race conditions that are hard to trigger during testing. Static analysis methods, in contrast, analyze the source code without executing it and generally offer faster results. Static analysis is advantageous in identifying race conditions that might not manifest during dynamic testing. With the massive number of threads available in the latest computing architectures, the interest in static analysis is growing to

complement dynamic analysis in performing race detection for modern systems.

Hybrid approaches that exploit both static and dynamic analyses become promising for discerning potential data races with increased fidelity. Nowadays, with the breakthrough in machine learning technology [10, 12], methodologies with machine learning have gained traction, leveraging pattern recognition in program behavior or applying LLMs to enable data race detection.

DataRaceBench (DRB) is an open-source benchmark suite methodically and quantitatively designed to evaluate data race detection tools. It is particularly oriented towards the context of OpenMP, a widely used parallel programming model for multithreaded applications. More specifically, DRB contains microbenchmark programs both with and without data races, which are either manually crafted, derived from actual scientific applications, or generated as automatic optimization variants.

Despite its effective labeling system for collected microbenchmarks, DRB lacks a structured dataset specifically designed for machine learning training and evaluation. There is a clear demand for a well-curated dataset comprising prompt-response pairs, which is crucial for the fine-tuning process of LLMs. Such a dataset, tailor-made for machine learning applications, could significantly enhance the performance and efficacy of data race detection methodologies.

3 APPROACH

In this section, we elaborate on our approach by combining two principal routes designed to exploit the capabilities of Large Language Models for novel tasks. First, we evaluate three strategies for data race detection with LLMs. Second, we fine-tune two open-source LLMs for data race detection and identification of data race variable pairs. As shown in Figure 1, our approach relies on the proposed dataset, DataRaceBench-ML. This section uses several popular Large Language Models, such as GPT, Llama2, and StarChat-beta.

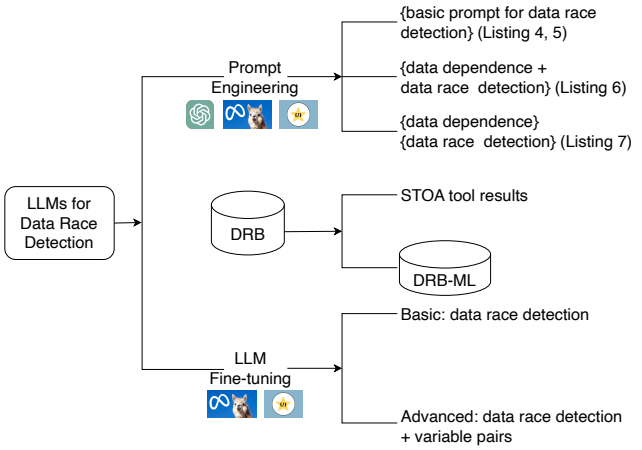

Figure 1: Overview of Our Approach for Data Race Detection using Large Language Models: GPT-3.5-turbo, GPT-4, Llama2, and StarChat-beta.

3.1 DataRaceBench-ML Dataset

The quality of datasets is essential for the success of any machine-learning approach. They determine the accuracy and robustness of a model's predictions. We processed the existing DataRaceBench V1.4.1 (DRB) to generate a new dataset, DataRaceBench-ML (DRB-ML), to facilitate efficient ML model training, fine-tuning, and evaluation. Each C/C++ microbenchmark from the DRB results in an entry in DRB-ML. Consequently, DRB-ML consists of 201 JSON files storing various key-value pairs - a direct correlation to the number of code snippets in the original DRB dataset.

Creating the DRB-ML dataset is a multi-step process. Firstly, we extract labels from each code snippet in the DRB dataset and store them in JSON. Table 1 illustrates the keys and their corresponding values in DRB-ML JSON files, each playing a crucial role in training ML models for data race detection. The 'data race-yes/no' label provides a binary indicator of data race conditions, while the 'var_pairs' label includes a list of variable pairs related to potential data races, along with their names, locations, and operation types ('w' for write or 'r' for read). Together, these labels offer a comprehensive view of the features our models need to analyze for effective data race detection. This step is carried out using scripts that are designed to sift through code comments and metadata to find relevant information. Listing 2 shows an example in DRB-ML labels derived from a microbenchmark presented in Listing 1 from DRB. We omit the code content to better represent the paper. It is worth mentioning that the "line" value in DRB-ML is based on the code without comments.

```c
/*
A loop with loop-carried anti-dependence.
Data race pair: a[i+1]@64:10:R vs. a[i]@64:5:W
*/
#include <stdio.h>
int main(int argc, char* argv[])
{
  int i;
  int len = 1000;

  int a[1000];

  for (i=0; i<len; i++)
    a[i]= i;

#pragma omp parallel for
  for (i=0;i< len -1 ;i++)
    a[i]=a[i+1]+1;

  printf ("a[500]=%d\n", a[500] );
  return 0;
}
```

Listing 1: DRB001-antidep1-orig-yes.c

```json
{
  "ID": "001",
  "name": "DRB001-antidep1-orig-yes.c",
  "DRB_code": "..."
  "data_race": 1,
  "trimmed_code": "..."
  "code_len": 262,
  "data_race_label": "Y1",
  "var_pairs":
  [
      "{
```

Table 1: Keys and Values in DRB-ML

Keys	Value type	Description
ID	int	A unique index number starting from 1.
name	str	The original filename of the DRB file.
DRB_code	str	The original code present in DRB microbenchmarks.
trimmed_code	str	The DRB_code with all comments removed.
code_len	int	An integer value representing the string length of the trimmed code.
data_race	int	The presence of a data race is indicated by 1, and its absence by 0.
data_race_label	str	This label indicates the type of data race condition (race-yes or race-no) that DRB marks.
var_pairs	[str]	This is a list of pairs of variables associated with a data race. It is empty when "data_race" is 0. Each item follows the format [VAR0, VAR1], where VAR1 depends on VAR0. Each variable is represented by a JSON string with keys in below.
pair["name"]	[str]	Variable names.
pair["line"]	[int]	Variable's line number in trimmed code.
pair["col"]	[int]	Variable's column number in trimmed code.
pair["operation"]	[str]	The operation performed on the variable. The value is either 'w' (representing a write operation) or 'r' (representing a read operation).

```
        "name": ["a[i+1]", "a[i]"],
        "line": [64, 64],
        "col": [10, 5],
        "operation": ["R", "W"]
      }"
  ]
}
```

Listing 2: DRB-ML-001.json

The second step involves the creation of a data template for the prompt-response pairs. The prompts are formulated to guide the LLM in identifying data races and to provide information about variables that might be causing them. The responses are simple labels indicating whether a data race exists or not.

In the final step, we employ scripts to pull the code and the information generated in the first step. The result is a structured prompt-response pair for each code in the DRB-ML dataset.

Upon completion of this process, each code snippet in the DRB-ML dataset contains three key pieces of information: the presence (or absence) of a data race, pairs of variables that could cause a data race, and the corresponding line numbers where these variables are found.

```
{
"prompt": """You are an HPC expert. Examine the following
     code and identify if there's a data race. If a data
     race is present, specify the variable pairs causing
     it, along with their line numbers and operations.
    Code: ...""",
"response": """Yes, the provided code exhibits data race
    issues. The data race is caused by the variable 'x'
    at line 9 and the variable 'x' at line 26. Both
    instances involve write operations."""
}
```

Listing 3: Prompt-response example for DRB-ML-193

3.2 Experiment Setup

Dataset: As outlined in Figure 1, we employ two strategies to evaluate the proficiency of LLMs in data race detection. First, we extract a subset of DRB-ML, ensuring that the data items have token sizes of less than 4k to accommodate the input sequence size limits of the selected LLMs. This sub-set consists of 198 out of the total 201 entries in DRB-ML. For the prompt engineering approach, we utilize the labels in the sub-set to assess the performance of the LLMs. Conversely, for fine-tuning the LLMs, we rely on the prompt-response pairs in DRB-ML for the fine-tuning process. The performance of the fine-tuned LLMs is then evaluated using the labels from the dataset.

Models: We start our experiments by employing four pre-trained large language models for data race detection. The chosen models, including GPT-3.5-turbo, GPT-4, Llama2-7b, and StarChat-beta with 16 Billion parameters, represent a variety of architectures and are reputed for their performance on a range of tasks.

3.3 Prompt Engineering for Data Race Detection

Prompt engineering is a key technique in harnessing the power of Large Language Models. It is a process wherein users tailor input prompts to elicit a particular response from the model. The goal is to craft prompts that effectively guide the model's responses in the desired direction. While it is hard to define the best prompt [34], a well-designed prompt can enable the model to provide insightful, precise, and contextually appropriate answers.

For the specific task of data race analysis using LLMs, we first delineated the expectations regarding their output in the following scenarios:

(1) **S1. Data Race Detection:** Given a code snippet, LLMs are expected to decisively and concisely determine the presence of a data race.
(2) **S2. Identification of Data Race Variables:** LLMs should analyze the code to identify the variables responsible for the data race.
(3) **S3. Details on Data Race-related Variables:** LLMs ought to disclose pertinent information concerning each involved variable, including its name, its line number in the code, and the specific operation (either read or write) performed on it.

With our goals outlined, we started with two basic prompts for data race detection. As an illustration, Listing 4 focuses on data race detection (S1), while Listing 5 instructs the LLMs to provide details on data race variables in conjunction with their data race detection findings (S1-3). Intriguingly, our preliminary experiments revealed a notable variance in the data race detection outcomes, shown in table 2, when comparing the responses generated from GPT-3.5-turbo with the two basic prompts.

```
"""
You are an expert in High-Performance Computing. Examine the code presented to you and ascertain if it contains any data races.
Begin with a concise response: either 'yes' for the presence of a data race or 'no' if absent.

{Code_to_analyze}
"""
```

Listing 4: Basic Prompt 1 (BP1) template

```
"""
You are an expert in High-Performance Computing. Examine the code presented to you and ascertain if it contains any data races.
Begin with a concise response: either 'yes' for the presence of a data race or 'no' if absent.
detail each occurrence of a data race by specifying the variable pairs involved, using the JSON format outlined below:
{
"name": Names of each pair of variables involved in a data race.
"line": line numbers of the paired variables within the code.
"col": column number of the paird variables with in their line.
"operation_types": Corresponding operations, 'W' for write operation and 'R' for read operation.
}

{Code_to_analyze}
"""
```

Listing 5: Basic Prompt 2 (BP2) template

Table 2: Data race detection results of GPT-3.5-turbo using basic prompts 1 (BP1) and basic prompts 2 (BP2) shown in listing 4 and 5, respectfully.

Prompts	TP	FP	TN	FN	Recall	Precision	F1
BP1	66	55	43	34	0.660	0.545	0.597
BP2	35	26	72	65	0.35	0.574	0.435

The findings showcased in Table 2 suggest that multi-task prompts necessitate meticulous crafting in contrast to their simpler, more concise counterparts. This observation aligns with prior research in prompt engineering, where "greedy" prompts yielded sub-optimal performance [34]. Given these insights, we opted to refine our prompt engineering for data race detection based on Listing 4 while addressing the tasks of S2 and S3 through the fine-tuning approach discussed in Section 3.4.

To enhance the quality of prompts for data race detection, we integrated insights from traditional tools and principles of concurrent programming. We crafted a prompt shown in Listing 6 to explicitly instruct the LLMs to look for instances where two or more threads are simultaneously accessing the same memory location without proper synchronization, and at least one access is a write operation. Our preliminary results in table 2 show that a simple and concise prompt may be more efficient. Therefore, we broke the instruction in Listing 6 into two prompts and executed them sequentially in a chat mode of the LLMs. This Chain-of-thoughts (COT) strategy introduced by Zhang et al. [32] facilitates step-by-step thinking before answering a question, making each step simple and concise.

```
"""
You are an expert in High-Performance Computing (HPC). Examine the provided code to identify any data races based on data dependence analysis.
For clarity, a data race occurs when two or more threads access the same memory location simultaneously in a conflicting manner, without sufficient synchronization, with at least one of these accesses involving a write operation. It's crucial to analyze data dependence before determining potential data races.
Begin with a concise response: either 'yes' for the presence of a data race or 'no' if absent.
```

Listing 6: Advanced Prompt 1 (AP1) for data race detection. AP1 extends BP1 by giving some details of data race detection including its definition and key analysis.

```
"""You are an expert in High-Performance Computing (HPC). Analyze data dependence in the given code.

{Code_to_analyze}
"""
```

Chain1 in AP2. Chain1 guides the LLMs to check the data dependence in the given code.

```
"""A data race occurs when two or more threads access the same memory location simultaneously in a conflicting manner, without sufficient synchronization, with at least one of these accesses involving a write operation. Identify any data races based on the given data dependence information.
Begin with a concise response: either 'yes' for the presence of a data race or 'no' if absent.
"""
```

Chain2 in AP2. With the output of Chain1 as a part of its input, Chain2 focuses on the data race detection task.

Listing 7: Advanced Prompt 2 (AP2). AP2 utilizes the chain-of-thoughts strategy to break AP1 into a chain. Chain1 and Chain2 are connected using LangChain's SequentialChain.

In summary, we employed various prompt engineering strategies for data race detection, referencing Listings 4, 5, 6, and 7.

3.4 LLM Fine-tuning for Data Race Analysis

Settings. The DRB-ML dataset, as detailed in Section 3.1, provides foundational prompt-response templates designed specifically for data race detection. Building on this, we crafted two distinct prompt-response sets from the DRB-ML templates: one for detecting data

races and another for identifying the associated variables. Our fine-tuning process follows prior works utilizing human feedback to enhance large language models [35].

We chose the Llama2-7b and StarChat-beta models as our candidate base models for fine-tuning. We employed PyTorch version 2.01 and DeepSpeed 0.9.5 to support fine-tuning. For the Llama2-7b model, we adopted a learning rate of 2e-4, set the maximum sequence length to 256, and used the Adam optimizer. Conversely, for the StarChat-beta model, all settings remained consistent except for a learning rate adjustment to 9.65e-6. We set the batch size to be 4 per GPU for training. To optimize memory usage during fine-tuning, we integrated QLoRA [7], setting the LoRA attention dimension to 64 and applying a dropout rate of 0.1. Our training process utilized the cross-entropy loss.

Fine-tuning objective. Three scenarios were introduced in Section 3.3 for data race analysis. In the fine-tuning approach, we set two objectives for LLM fine-tuning: First, LLM fine-tuning for data race detection. And second, LLM fine-tuning for data race variable identification.

Fine-tuning dataset. Using the DRB-ML dataset, we utilized labels in the DRB-ML dataset to create two sets of 198 prompt-response pairs for data race detection and variable identification.

- Listing 8 shows an instance of prompt-response pairs derived from Listing 4 for LLM fine-tuning for basic data race detection.
- Listing 9 shows an instance of prompt-response pairs derived from LLM fine-tuning for advanced data race detection with variable identification.

```
{
"prompt":
"""
You are an expert in High-Performance Computing. Examine
    the code presented to you and ascertain if it
    contains any data races.
Begin with a concise response: either "yes" for the
    presence of a data race or "no" if absent.

{Code_to_analyze}
""",
"response": """yes"""

}
```

Listing 8: Instance of basic fine-tuning (basic-FT) prompt-response pairs for data race detection.

```
{
"prompt":
"""
You are an expert in High-Performance Computing. Examine
    the code presented to you and ascertain if it
    contains any data races.
Detail each occurrence of a data race by specifying the
    variable pairs involved using the JSON format
    outlined below:
{
"variable_names": Names of each pair of variables
    involved in a data race.
"variable_locations": line numbers of the paired
    variables within the code.
"operation_types": Corresponding operations, either '
    write' or 'read'.
}
{}
""",
"response":
"""
"yes",
{
    "data_race": 1,
    "variable_names": ["a[i]", "a[i+1]"],
    "variable_locations": [14, 14],
    "operation_types": ["write", "read"]
}
"""
}
```

Listing 9: Instance of advanced fine-tuning (advanced-FT) prompt-response pairs for advanced data race detection

3.5 Five-fold Crossing Validation

We implemented a stratified k-fold cross-validation approach with $k = 5$ to accomplish an unbiased evaluation. This method is designed to retain a consistent proportion of positive to negative samples in each fold, mirroring the overall dataset's structure.

The subset of DRB-ML used in our work showcases a distribution of roughly 50.5% positive(data race-yes) cases and 49.5% negative(data race-no) cases. In crafting the 5-fold cross-validation, each fold is meticulously constructed to emulate this distribution. This delineation averages out to each fold, accommodating about 20 positive cases and 19.6 negative cases.

Given the indivisibility of data points in a practical setting, the allocation was determined as follows: Three of the folds were populated with both 20 positive and 20 negative cases, making up 40 data points in each of these folds. The remaining two folds were assembled with 20 positive cases and 19 negative cases each, resulting in 39 data points for each of these folds. By adopting this stratified 5-fold cross-validation, we provide a representative sample in each partition, ensuring a comprehensive and robust evaluation of LLMs.

3.6 Evaluation Metrics

In our study, we assess the performance of Large Language Models (LLMs) by examining their outputs in the context of three scenarios, as detailed in Section 3.3. These scenarios—S1, S2, and S3—serve as binary classification tasks, allowing us to compute the counts of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) by comparing the LLM outputs with the ground truth.

To quantify the performance of the LLMs, we utilize established metrics such as recall (R), precision (P), and the F1 score (F1). Additionally, we compute the average value (AVG) and standard deviation (SD) for these metrics across 5-fold cross-validation experiments.

It's worth noting that the values for recall, precision, and F1 score—as well as their respective averages—range from 0 to 1, with higher values indicating better performance. Additionally, a lower standard deviation is indicative of more consistent performance across the different folds of validation, making a lower SD score preferable.

4 EXPERIMENTAL RESULTS

This section presents the outcomes of our experiments, encompassing both prompt engineering and model fine-tuning exercises.

4.1 Prompt Engineering for Data Race Detection

Leveraging the trimmed code snippets from the DRB-ML dataset, we produced a set of three prompts for every code following the three prompt engineering strategies discussed in Section 3.3. As suggested by the results in Table 2, we did not adopt BP2 because the "greedy" prompts yield sub-optimal performance.

- **BP1:** Based on the template from Listing 4. This prompt is succinct, directing LLMs straightforwardly to detect data races.
- **AP1:** Derived from Listing 6, this prompt instructs LLMs to emulate traditional tool methodologies, emphasizing data dependence analysis prior to ascertaining potential data races.
- **AP2:** Adopting the template from Listing 7, this variant separates the dual steps from AP1, adhering to a Chain-of-Thought design approach.

Subsequent to the model's output generation, we transformed these outputs into prediction labels. These predictions were then evaluated against the definitive "data_race" labels found within DRB-ML. Comprehensive results of this assessment can be found in Table 3, where values in **bold** signify the best performance across all tools, while values in red denote the top-performing LLM.

Table 3: Comparison of a representative traditional tool, Intel Inspector, and four LLMs: GPT-3.5-turbo, GPT-4, StarChat-beta, and Llama2-7b. We use three prompts: BP1, AP1, and AP2 to check if given codes contain data race. Values in bold signify the best performance across all tools, while values in green denote the top-performing LLM.

Model	Prompt	TP	FP	TN	FN	R	P	F1
Inspector	N/A	88	44	53	11	**0.889**	0.667	**0.762**
GPT3	BP1	66	55	43	34	0.660	0.545	0.597
	AP1	63	56	42	37	0.630	0.529	0.575
	AP2	69	54	44	31	0.690	0.561	0.619
GPT4	BP1	77	28	70	23	0.770	0.733	0.751
	AP1	78	30	68	22	0.780	0.722	0.750
	AP2	78	28	68	22	0.780	**0.736**	0.757
StarChat	BP1	63	68	30	37	0.630	0.481	0.545
	AP1	62	67	31	38	0.620	0.481	0.541
	AP2	63	61	37	37	0.630	0.508	0.563
Llama	BP1	65	57	41	35	0.650	0.533	0.586
	AP1	65	57	41	35	0.650	0.533	0.586
	AP2	66	55	43	34	0.660	0.545	0.597

4.2 Basic LLM Fine-tuning: Data Race Detection

To the best of our understanding, neither the training dataset for LLama2-7b nor StarChat-beta incorporates the DataRaceBench data upon reviewing their source. As such, we adopted the 5-fold cross-validation methodology detailed in Section 3.5 to fine-tune these open-source LLMs. We employed the basic prompt-response (basic-FT) pairs throughout the fine-tuning and validation phases, as exemplified in Listing 8.

Table 4 presents the results from this 5-fold cross-validation for the fine-tuned StarChat-beta and LLama2-7b models. The up-arrow indicates the performance increase by fine-tuned models compared with the original pre-trained versions.

Broadly speaking, the fine-tuned models demonstrated enhanced F1 score and consistency performance. The StarChat-beta model registered improvements across nearly all metrics for data race detection, with the sole exception being recall consistency. Conversely, while the Llama2-7b model saw a dip in its recall score, it exhibited advancements in other evaluation metrics.

Table 4: Average (AVG) and Standard Deviation (SD) of Recall, Precision, and F1 Score from a 5-fold cross-validation for data race detection using StarChat-beta, Llama2-7b, and their fine-tuned (FT) models with basic-FT prompts. Green indicates improved performance with fine-tuned models, while red signifies decreased performance.

Model	AVG of R	SD of R	AVG of P	SD of P	AVG of F1	SD of F1
StarChat	0.630	**0.045**	0.482	0.041	0.546	0.039
StarChat-FT	**0.670**	0.057	0.541	**0.037**	**0.598**	**0.038**
Llama	0.650	0.137	0.532	0.094	0.584	0.109
Llama-FT	0.640	0.082	**0.543**	0.054	0.586	0.061

4.3 Advanced LLM Fine-tuning: Data Race Detection and Data Race Variable Identification

As highlighted in the approach section, identifying data race-related variable pairs and extracting their detailed information poses significant challenges. Initially, we assessed the LLMs' performance concerning data race variable identification. Subsequently, we specifically fine-tuned the StarChat-beta and Llama2-7 models for this task.

Table 5: Comparison of results of advanced data race detection with variable identification, using four LLMs. Values in bold signify the best performance across all models.

Model	TP	FP	TN	FN	Recall	Precision	F1
GPT3	12	54	44	88	0.120	0.182	0.145
GPT4	14	31	67	86	**0.140**	**0.311**	**0.193**
StarChat	7	66	32	93	0.070	0.096	0.081
Llama	5	65	33	95	0.050	0.071	0.059

Table 5 showcases the performance metrics of the selected models before fine-tuning, while Table 6 showcases the results from the 5-fold cross-validation. The fine-tuned StarChat-beta and LLama2-7b models are compared to their original pre-trained versions. We consistently employed the advanced-FT prompt-response pairs throughout the fine-tuning and validation stages, as depicted in Listing 9. Although the performance of the StarChat-beta model improved after fine-tuning, this enhancement came with an added inconsistency. Conversely, the Llama2-7b model didn't exhibit any significant improvements, potentially due to the limited training dataset.

Table 6: Average (AVG) and Standard Deviation (SD) Recall, Precision, and F1 score of the 5-fold crossing validation for the advanced data race variable identification with StarChat-beta, Llama2-7b, and the fine-tuned (FT) models. Green indicates improved performance with fine-tuned models, while red signifies decreased performance.

Model	AVG of R	SD of R	AVG of P	SD of P	AVG of F1	SD of F1
StarChat	**0.070**	**0.045**	0.096	**0.063**	0.081	**0.052**
StarChat-FT	**0.070**	0.057	0.103	0.087	**0.083**	0.069
Llama	0.050	0.050	**0.085**	0.087	0.063	0.064
Llama-FT	0.050	0.050	0.092	0.086	0.064	0.063

4.4 Observation

Through meticulously crafted experiments focused on data race analysis using LLMs, we derived the following insights from our results:

- In general, GPT-4 stands out as the premier pre-trained model for data race analysis, excelling particularly in identifying data race-related variables. Nevertheless, the open-source models, namely StarChat-beta and Llama2-7b, demonstrate significant potential. With the right fine-tuning, they could indeed surpass the GPT series in data race detection capabilities.
- While traditional tools achieve superior performance in terms of the F1 score when compared to LLMs, testing with the DataRaceBench data indicates that GPT-4 exhibits noteworthy potential. This is impressive, given that GPT-4 is designed for general-purpose tasks and not specifically optimized for this domain.
- Our initial results, showcased in Table 2, indicate a clear trend: simple and concise prompts yield better results by LLMs. Our extensive prompt engineering results reinforce this observation, as presented in Table 3. Specifically, with the exception of the Llama2-7b model, all other models displayed enhanced performance with 'BP1'—a succinct prompt, as compared to 'BP2'—a multi-task oriented prompt, when it came to data race detection.
- Our results from fine-tuning demonstrate the potential of open-source LLMs in handling data race analysis tasks.

4.5 Challenges and Possible Solutions

In our exploration of data race analysis with LLMs, spanning from dataset preparation to LLM inference, fine-tuning, and evaluation, we encountered several challenges:

- **Dataset:** The dataset preparation was both time-consuming and labor-intensive, further complicated by the scarcity of available datasets. This scarcity subsequently affected the efficacy of LLM fine-tuning. Potential remedies include:
  - Crawling data from open-source repositories.
  - Generating synthetic datasets tailored for training.
  - Automating the dataset processing stages using LLMs.
- **Natural Language Output Processing:** As text generation models, LLMs produce outputs in natural language. Parsing and processing these outputs present considerable challenges. One approach to mitigating this challenge is by directing LLMs to adhere to specific output formats. Initially, our DRB-ML dataset's prompt-response pairs, as exemplified in Listing 3, contained natural language outputs. We later transitioned to structured JSON outputs, as depicted in Listing 5. Nonetheless, not every LLM consistently maintains designated output formats, leading us to employ regular expressions for parsing.
- **General Challenges for PLP with LLMs:** Although LLMs have achieved great success in many areas, their success happens mostly in NLP tasks. The processes of training data collection, tokenization, and embedding representations for the LLMs are all finely tuned to cater to the requirements of NLP applications. Advancements in LLMs have recently incorporated programming language source codes and language-specific content into their training datasets. However, the quality of this training data remains suboptimal. A notable issue is the inclusion of incomplete or incorrect code snippets that standard compilers cannot successfully compile. This deficiency has drawn our attention, highlighting the pressing need to enhance the quality of training data for Programmable Language Models in the context of programming tasks. Addressing this challenge is imperative to fully empower LLMs to effectively perform PLP tasks.

5 CONCLUSION

In this paper, we have explored the capabilities of large language models for the task of detecting data races in OpenMP programs. A dedicated dataset, DRB-ML, was created based on DataRaceBench to evaluate and fine-tune LLMs. The results show that LLMs have the potential to become an alternative solution for data race detection. However, they cannot outperform traditional data race detection tools without improved training datasets or novel code representations that capture more code semantics.

In the future, we are interested in expanding DRB-ML to include more data items using data scraping and augmentation techniques. We will also explore different modalities beyond text as input, such as abstract syntax trees, dependence graphs, and control-flow graphs.

ACKNOWLEDGMENTS

Prepared by LLNL under Contract DE-AC52-07NA27344(LLNL-CONF-853160) 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] Vamshi Basupalli, Tomofumi Yuki, Sanjay Rajopadhye, Antoine Morvan, Steven Derrien, Patrice Quinton, and David Wonnacott. 2011. ompVerify: polyhedral analysis for the OpenMP programmer. In *OpenMP in the Petascale Era: 7th International Workshop on OpenMP, IWOMP 2011, Chicago, IL, USA, June 13-15, 2011. Proceedings 7*. Springer, 37–53.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] Le Chen, Pei-Hung Lin, Tristan Vanderbruggen, Chunhua Liao, Murali Emani, and Bronis de Supinski. 2023. LM4HPC: Towards Effective Language Model Application in High-Performance Computing. *arXiv preprint arXiv:2306.14979* (2023).

[4] Le Chen, Quazi Ishtiaque Mahmud, and Ali Jannesari. 2022. Multi-View Learning for Parallelism Discovery of Sequential Programs. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 295–303.

[5] Le Chen, Quazi Ishtiaque Mahmud, Hung Phan, Nesreen Ahmed, and Ali Jannesari. 2023. Learning to Parallelize with OpenMP by Augmented Heterogeneous

AST Representation. *Proceedings of Machine Learning and Systems* 5 (2023).
[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
[7] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314* (2023).
[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[9] Xianzhong Ding, Le Chen, Murali Emani, Chunhua Liao, Pei-Hung Lin, Tristan Vanderbruggen, Zhen Xie, Alberto E. Cerpa, and Wan Du. 2023. HPC-GPT: Integrating Large Language Model for High-Performance Computing. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023)*.
[10] Xianzhong Ding and Wan Du. 2022. Drlic: Deep reinforcement learning for irrigation control. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 41–53.
[11] Xianzhong Ding, Wan Du, and Alberto Cerpa. 2019. OCTOPUS: Deep reinforcement learning for holistic smart building control. In *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*. 326–335.
[12] Xianzhong Ding, Wan Du, and Alberto E Cerpa. 2020. Mb2c: Model-based deep reinforcement learning for multi-zone building control. In *Proceedings of the 7th ACM international conference on systems for energy-efficient buildings, cities, and transportation*. 50–59.
[13] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).
[14] Patrick Flynn, Tristan Vanderbruggen, Chunhua Liao, Pei-Hung Lin, Murali Emani, and Xipeng Shen. 2022. Finding Reusable Machine Learning Components to Build Programming Language Processing Pipelines. *arXiv preprint arXiv:2208.05596* (2022).
[15] Intel. [n. d.]. *Inspector*. https://www.intel.com/content/www/us/en/developer/tools/oneapi/inspector.html
[16] Marzena Karpinska and Mohit Iyyer. 2023. Large language models effectively leverage document-level context for literary translation, but critical errors persist. *arXiv preprint arXiv:2304.03245* (2023).
[17] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and Individual Differences* 103 (2023), 102274.
[18] Bin Lei, Caiwen Ding, Le Chen, Pei-Hung Lin, and Chunhua Liao. 2023. Creating a Dataset Supporting Translation Between OpenMP Fortran and C++ Code. *arXiv preprint arXiv:2307.07686* (2023).
[19] Bin Lei, Chunhua Liao, Caiwen Ding, et al. 2023. Boosting Logical Reasoning in Large Language Models through a New Framework: The Graph of Thought. *arXiv preprint arXiv:2308.08614* (2023).
[20] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023).
[21] Chunhua Liao, Pei-Hung Lin, Joshua Asplund, Markus Schordan, and Ian Karlin. 2017. 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*. 1–14.
[22] LLVM. 2023. THREADSANITIZER. https://clang.llvm.org/docs/ThreadSanitizer.html
[23] OpenAI. 2023. GPT-4 Technical Report. *ArXiv* abs/2303.08774 (2023).
[24] Polyvios Pratikakis, Jeffrey S Foster, and Michael Hicks. 2006. Locksmith: context-sensitive correlation analysis for race detection. *Acm Sigplan Notices* 41, 6 (2006), 320–331.
[25] Bo Shen, Jiaxin Zhang, Taihong Chen, Daoguang Zan, Bing Geng, An Fu, Muhan Zeng, Ailun Yu, Jichuan Ji, Jingyang Zhao, et al. 2023. PanGu-Coder2: Boosting Large Language Models for Code with Ranking Feedback. *arXiv preprint arXiv:2307.14936* (2023).
[26] Ali TehraniJamsaz, Mohammed Khaleel, Reza Akbari, and Ali Jannesari. 2021. Deeprace: A learning-based data race detector. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 226–233.
[27] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
[28] Jan Wen Voung, Ranjit Jhala, and Sorin Lerner. 2007. RELAY: static race detection on millions of lines of code. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 205–214.
[29] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).
[30] Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. 2023. Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 7443–7464.
[31] Wenxuan Zhang, Yue Deng, Bing Liu, Sinno Jialin Pan, and Lidong Bing. 2023. Sentiment Analysis in the Era of Large Language Models: A Reality Check. *arXiv preprint arXiv:2305.15005* (2023).
[32] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2022. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493* (2022).
[33] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).
[34] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022).
[35] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593* (2019).