



# LLM4VV: Developing LLM-Driven Testsuite for Compiler Validation

Christian Munley, Aaron Jarmusch, Sunita Chandrasekaran  
Contact us: schandra@udel.edu



Computational Research & Programming Lab (CRPL)  
Department of Computer & Information Sciences

## Background

**OpenACC:** A directive-based parallel programming model designed to accelerate application on GPUs and other devices. It abstracts hardware complexities and enables portability across different high-performance computing (HPC) architectures.

**Large Language Models (LLMs):** Artificial intelligence models capable of understanding and generating natural language. Growing proficiency in code generation and with the potential to revolutionize software development.

**Compiler Validation:** Essential for ensuring that compilers correctly implement programming paradigms such as OpenACC. It involves rigorous testing to detect various types of issues, ensuring reliable software performance across diverse platforms.

**Research Gap:** Despite LLMs' growing skill in code generation, their application in compiler validation, such as in the context of parallel programming paradigms like OpenACC, is previously unexplored. Existing methods for compiler validation can be labor-intensive and error-prone, leading to potential significant errors in scientific research.

This work aims to bridge this gap by leveraging and improving the capabilities of state-of-the-art LLMs, including OpenAI's GPT-4 and Meta AI's Codellama, to automate generation of tests.

## Validation & Verification Motivation

Create validation and verification tests for parallel programming compilers from the standard specification

Revealing ambiguities in the OpenACC & OpenMP Specification

Determining missing implementation of a feature

Highlighting unmentioned restriction of a feature

Evaluating implementations for multiple target platforms

Identifying and reporting compiler bugs

This material is based upon work supported by NSF under grant no. 1814609 and support from OpenACC organization. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC ECR00029463. This research was supported in part through the use of DARWIN computing system supported by NSF under grant no. 191983.

## Methods

### Retrieval-Augmented Generation (RAG):

context retrieved from spec with vector database at inference, or via dictionary retrieval

**In-Context Learning:** examples of good tests shown in the prompt

**Code Template:** to standardize output for comparison

**Expressive Prompting:** in-depth instruction of the task

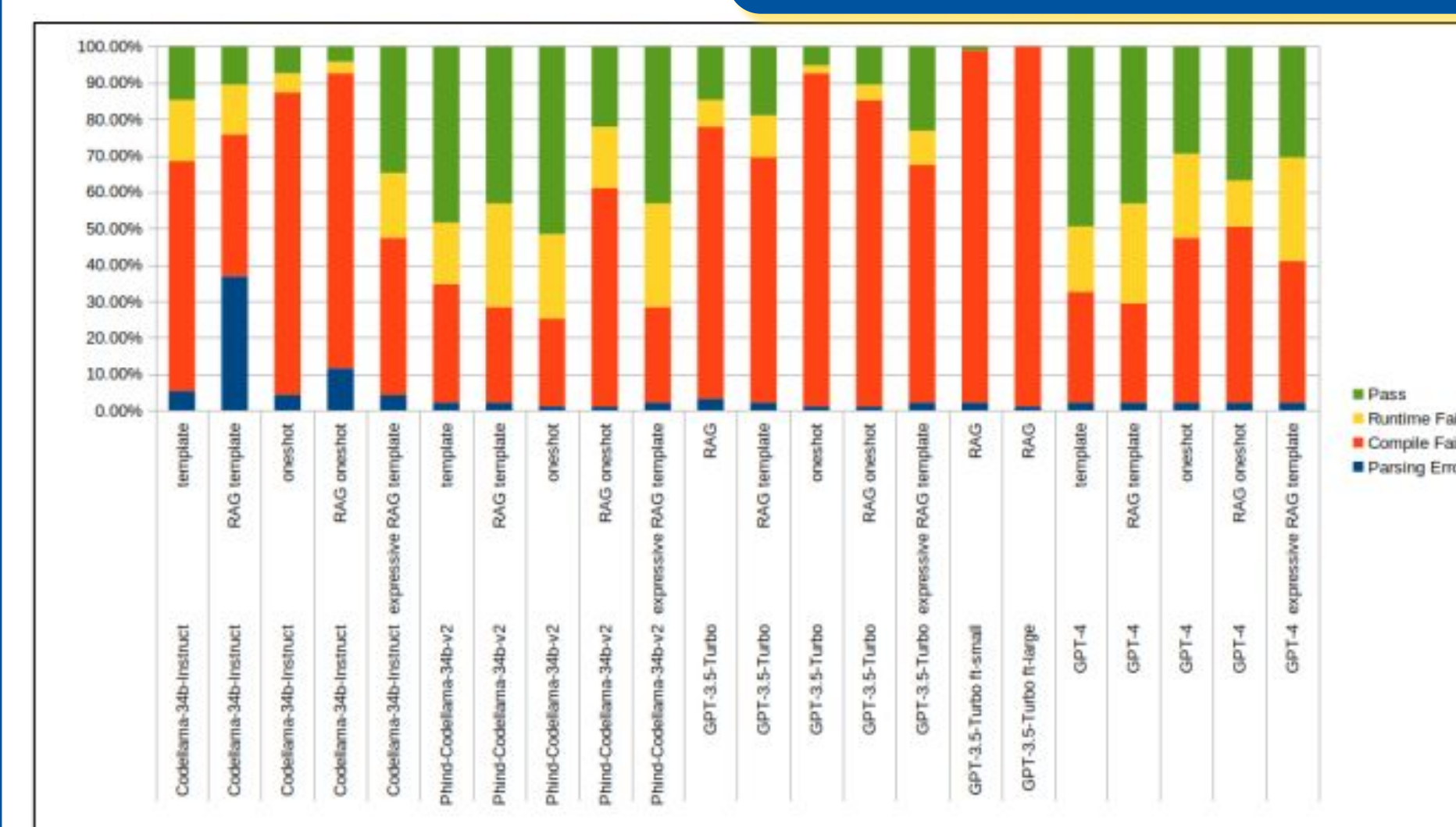
**Fine-Tuning:** training LLMs on task-specific dataset, e.g. our manually written testsuite

**LLMs:** GPT-3.5/4, Codellama-34b, fine-tuned variants

**Prompts:** Derived from spec. table of contents, permutations of clauses with compute constructs, and languages C/C++/Fortran

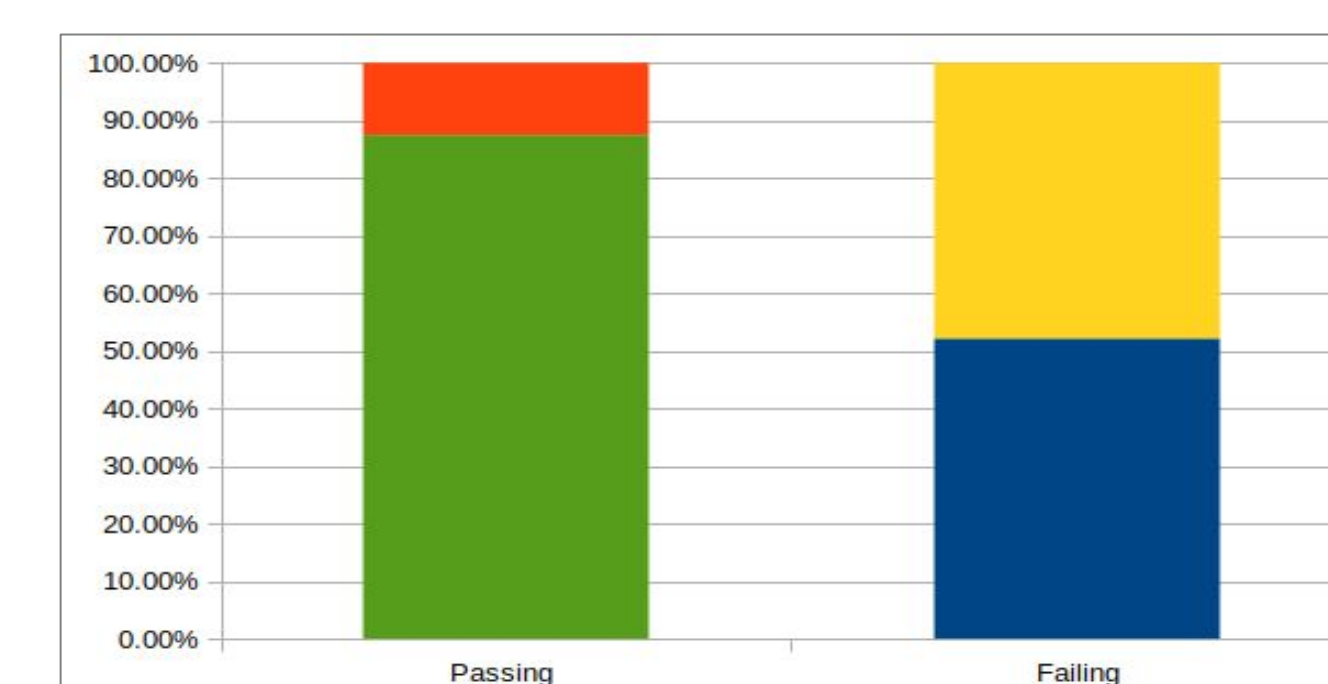
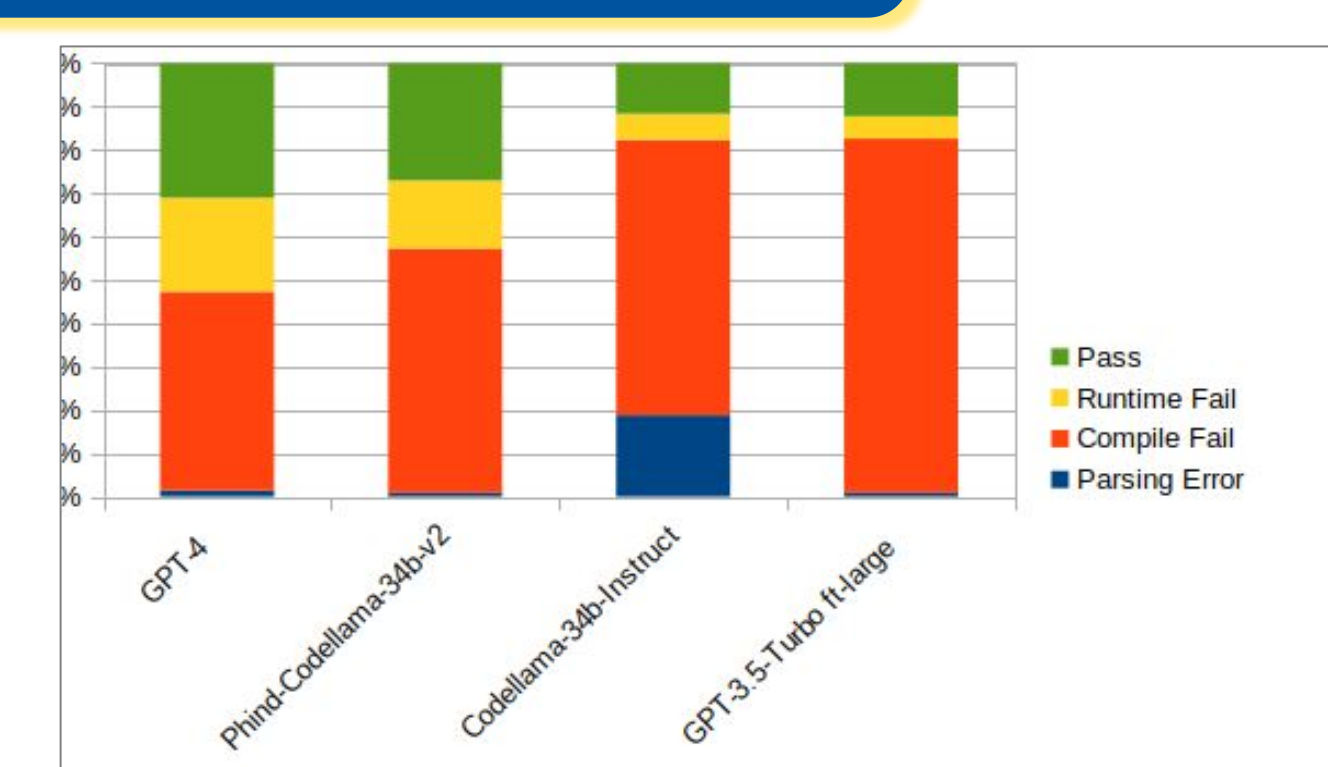
**Evaluation:** compare methods and LLMs; manual analysis of output: pass/fail, correct/incorrect openacc or base language.

## Results



The majority of passing tests are correct tests, while tests fail for a variety of reasons: Error in usage of OpenACC, Serial C Code, or other reasons.

GPT-4 using a code template and retrieval augmented generation outperformed all other configurations



## Challenges

- Addressing GPU memory limits during test generation. (peft, lora, quantization)
- Filtering out "junk output" from LLMs to maintain test quality. (parsing techniques)
- Overcoming initial fine-tuning hurdles for V&V tasks. (poor quality output)
- RAG difficulties with technical documents versus manual context extraction. (chunking by structure, RAG methods,)
- Lack of automated metrics for V&V tests, necessitating human analysis. (stage 3)
- Absence of established benchmarks for comparison or LLM selection.

## Implementation

### LLM Fine-tuning:

NERSC Perlmutter (4-8)x100 NVIDIA GPUs  
HuggingFace transformers and PEFT,  
PyTorch fully sharded data parallel,  
DeepSpeed ZeRO-3 memory optimization,  
fine-tuned on OpenACC V&V testsuite.  
OpenAI API with prompt-completion pair dataset.

### Testsuite infrastructure:

Infrastructure built in python to run generated tests with choice of compiler and hardware, and format results into JSON.  
Ran on Darwin V100 GPU

### LLM Inference:

DARWIN 4xV100 NVIDIA GPUs  
Meta AI Codellama inference script, HuggingFace transformers, OpenAI API  
Over 30 test suites generated, each taking ~5 hours on 4xV100 to complete.

## Example

### Prompt

Write a code in C to verify compiler implementation of the OpenACC specification of kernels construct reduction clause. Make sure to follow the template of the format provided. Include the provided header file, and any other necessary libraries. Write simple code to test kernels construct reduction clause. Try to isolate that feature while still using it correctly. This code is part of a testsuite that will be ran, so write complete code, don't leave it unfinished. The goal is to return 0 if the target feature, kernels construct reduction clause, is working properly, and not zero otherwise. The context below is from the most recent OpenACC specification, make sure to refer to it to produce up to date code.

{context from specification (RAG)}  
{code template / one-shot example}

## Response: "

### Generated Test

```
1 #include "acc_testsuite.h"
2 #define SIZE 100
3 #define SEED 12345
4
5 #ifdef T1
6 /*T1: kernels construct reduction clause, V:2.0-2.7*/
7 int test1(){
8     int i, err = 0;
9     int reduction_result = 0;
10    int a[SIZE];
11
12    srand(SEED);
13
14    // Initialize array with random numbers
15    for(i=0; i<SIZE; i++)
16        a[i] = rand()%100;
17
18    // OpenACC kernels construct with reduction clause
19    #pragma acc kernels copy(a[0:SIZE]) reduction(+:reduction_result)
20    for(i=0; i<SIZE; i++)
21        reduction_result += a[i];
22
23
24    // Verification
25    int expected_result = 0;
26    for(i=0; i<SIZE; i++)
27        expected_result += a[i];
28
29    if(reduction_result != expected_result){
30        err = 1;
31    }
32
33    return err;
34 }
```

## Conclusion

Developers implementing validation and verification testsuites can be greatly aided by automation. Instead of manually writing 1000s of tests, an experienced developer can generate tests, and approve or edit them.

We expect validation and verification efforts to trend towards automation as specifications constantly evolve and current developers find new projects.

### Current areas of development:

- Fine-tuning on more HPC data
- Implementing reinforcement-learning with human (or compiler) feedback
- Integrating HPC-tokenizer, improved RAG, prompt-tuning, etc.
- Developing benchmark for LLM V&V

### Resources:

Generated testsuites and results: <https://github.com/chrismun/LLM4VV>  
Fine-tuning dataset: <https://huggingface.co/datasets/chrismun/oaccv>  
Manual Testsuite - <https://github.com/OpenACCUserGroup/OpenACCv-V>  
CRPL OpenACCv&V Website - <https://crpl.cis.udel.edu/oaccv/>  
Preprint: <https://arxiv.org/pdf/2310.04963.pdf>