From Legacy to Portable: An Agentic AI Workflow for Fortran Code Translation and Cross-Architecture Optimization

Sparsh Gupta, Kamalavasan Kamalakkannan, Maxim Moraru, Galen Shipman, Patrick Diehl

SC25: International Conference for High Performance Computing, Networking, Storage and Analysis; St. Louis, MO, USA; November 16–21, 2025

Motivation

- **Fortran** remains foundational to many scientific applications and HPC codes across national labs and universities.
- Modern HPC systems are **GPU-accelerated heterogeneous architectures**, with GPUs from NVIDIA, AMD, etc.; each requiring vendor-specific frameworks (like CUDA, HIP, etc.) for efficient performance.
- **Kokkos**, developed at U.S. DOE National Labs, offers a single-source C++ abstraction to target multiple backends for performance portability.
- Can an **agentic AI workflow**, leveraging LLM "agents", automate this translation from Fortran to Kokkos to generate portable, performant HPC code?

Introduction

- Manually translating legacy Fortran code to programming models compatible with today's heterogeneous HPC systems is complex, error-prone, and labor-intensive.
- Porting Fortran kernels to Kokkos offers a flexible C++ based ecosystem to target multicore CPUs, AMD GPUs, NVIDIA GPUs, etc.
- To automate this process, we introduce an agentic AI workflow where specialized LLM agents, utilizing OpenAI models or open-source LLMs, work together to fully automate each pipeline stage.
- We evaluated this pipeline on five benchmark Fortran kernels: Conjugate Gradient (CG), Embarrassingly Parallel (EP), Multi-Grid (MG), Fourier Transform (FT) from the NAS Parallel Benchmarks, and DGEMM from OpenBLAS.

Tools

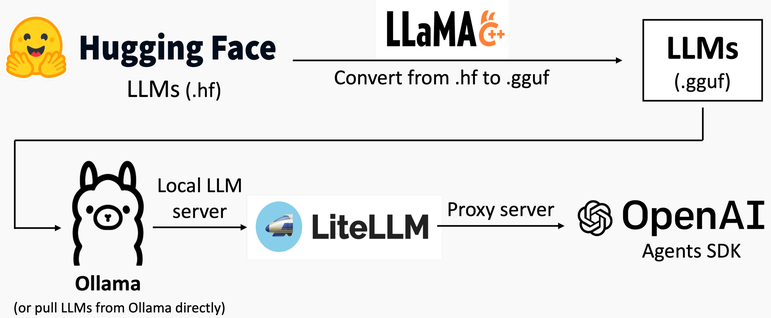

Figure: Deployment framework for open-source LLMs used in our workflow.

- **LLMs:** OpenAI models accessed using their API and open-source GGUF LLMs served via Hugging Face or Ollama, routed through a LiteLLM proxy on HPC clusters.
- **OpenAI Agents SDK:** A Python framework to build multi-agent workflows.
- **Spack** for multi-architecture environments; **SLURM** for job orchestration.

Methods

- A **Translator** agent converts the Fortran kernel to standalone C++/Kokkos code.
- **Validator** and **Fixer** agents ensure the code is syntactically correct and compilation-ready.
- **Build** and **Run** agents submit SLURM jobs to compile the code, collect runtime data, and execute performance sweeps.
- If failures occur, **Compile/Runtime Error Fixer** agents use an **Error Summarizer** agent to parse job logs, identify root causes, and patch the code iteratively.
- **Functionality Tester** verifies Kokkos output against Fortran output for standard test inputs; if mismatches occur, the **Functionality Fixer** attempts to resolve correctness.
- **Optimizer** agent analyzes performance from GPU profilers and proposes refinements.
- The workflow cycle repeats - translate, validate and/or fix, build, run, fix errors, optimize - until a high-performance, portable implementation is achieved.

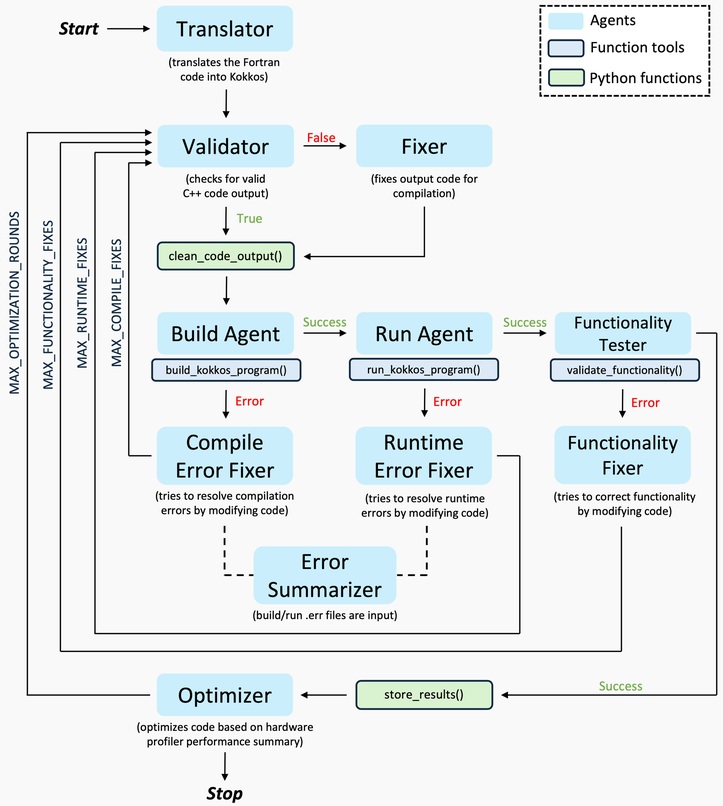

Figure: Agentic AI workflow to translate, validate, compile, run, test, debug, and optimize Fortran-to-Kokkos programs.

Results

- **LLM Benchmarking:** OpenAI models execute the entire pipeline successfully whereas Llama 4 Maverick often failed to converge within the fix thresholds.

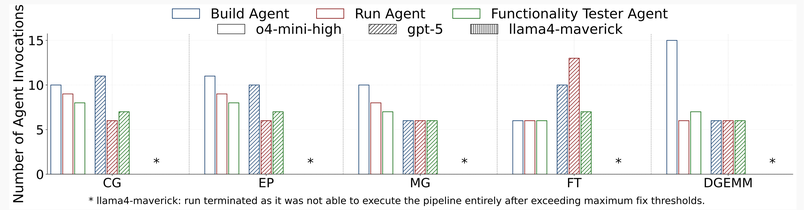

Figure: Total agent invocations per kernel for an entire successful run on AMD MI250.

- **Translation cost:** Achieved fully autonomous Fortran-to-Kokkos translation using paid models for under US$3.5 per kernel (ignoring GPU runtime costs).

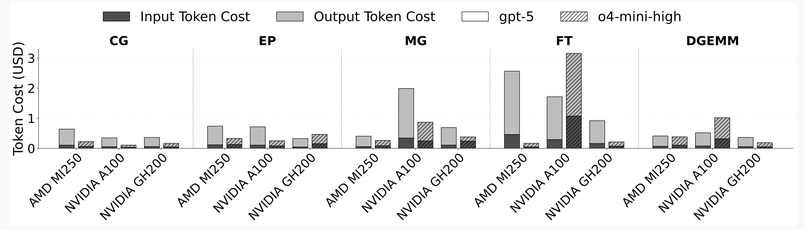

Figure: Total token cost (USD) by kernel and partition comparing GPT-5 vs o4-mini-high.

- **Optimization gains:** Agent-driven optimization consistently improved GFLOPS after the baseline version (v1).

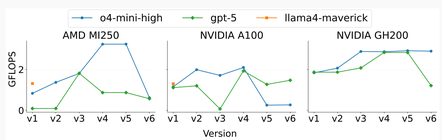

Figure: CG: GFLOPS at max input size (MAX_N) vs version across partitions/models.

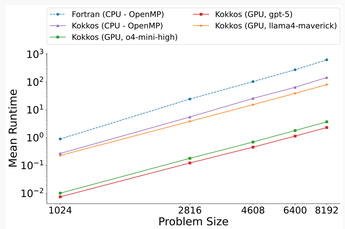

Figure: DGEMM runtime comparison (with most optimized versions for Kokkos) on NVIDIA A100.

Conclusion & Future Work

- **Conclusion:** This workflow demonstrates a proof of concept that legacy HPC benchmarks can be fully translated from Fortran to Kokkos and iteratively optimized to achieve better performance using paid LLMs in just hours for only a few US dollars.
- **Future Work:** Extend functionality testing beyond benchmarks by developing dynamic, LLM-driven correctness verification; explore heterogeneous agent setups that leverage different models for different agents (e.g., reasoning vs. coding).

Research presented in this poster was supported by the National Security Education Center (NSEC) Informational Science and Technology Institute (ISTI) using the Laboratory Directed Research and Development program of Los Alamos National Laboratory project number 20240479CR-IST.
This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.
This work was also supported by the U.S. Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001).

LA-UR-25-28074