
OAK RIDGE National Laboratory

SC25 ST. LOUIS NOV 16–21

# ChatHPC:
## Building the foundation of an AI assisted and productive HPC ecosystem









**PRESENTED BY**

Pedro Valero-Lara

PhD. Sr Computer Scientist <valerolarap@ornl.gov>



U.S. DEPARTMENT OF ENERGY

ORNL IS MANAGED BY UT-BATTELLE LLC
FOR THE US DEPARTMENT OF ENERGY

# ChatHPC, What is that?

## Infrastructure:

- A CLI Python library on top of:
  - Code Llama, PyTorch, and LoRA
- The ChatHPC library makes:
  - Fine-tuning, Testing, Refinement, Merge, and Inference

## Ecosystem:

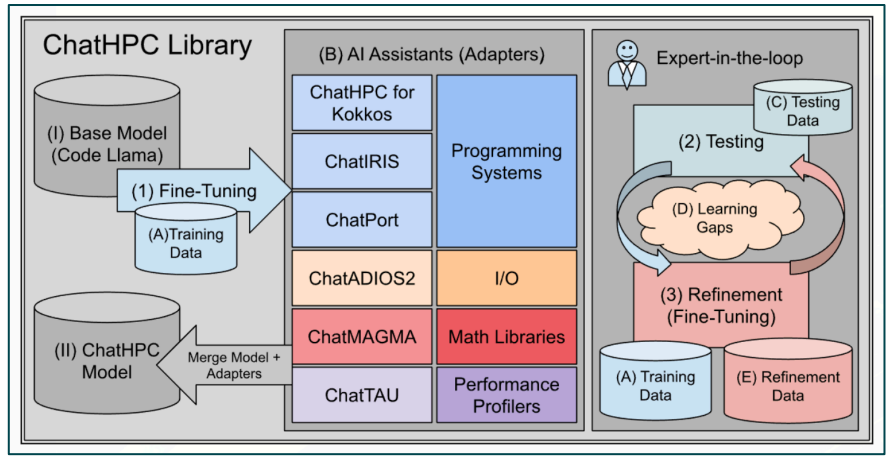
- A collection of AI assistants leveraging existing HPC efforts for multiple domains:
  - Programming Models, Math Libraries, I/O, Tooling, etc.
- And priorities:
  - Parallelization, Portability, Optimization, Evaluation, etc.

ChatHPC CLI:

```
$ chathpc train   # Finetune the assistent.
$ chathpc verify  # Verify the assistent on training set.
$ chathpc test    # Test the asssistent on unseen data.
$ chathpc run     # Interactivly run the assistent.
```

Interactive Run Session:

```
$ chathpc ()> /context
Context: Introduction to Kokkos
$ chathpc (Introduction to Kokkos)> What is LayoutLeft?
LayoutLeft refers column-major layout where consecutive entries in
      the same column of a 2-D array are contiguous in memory.
```

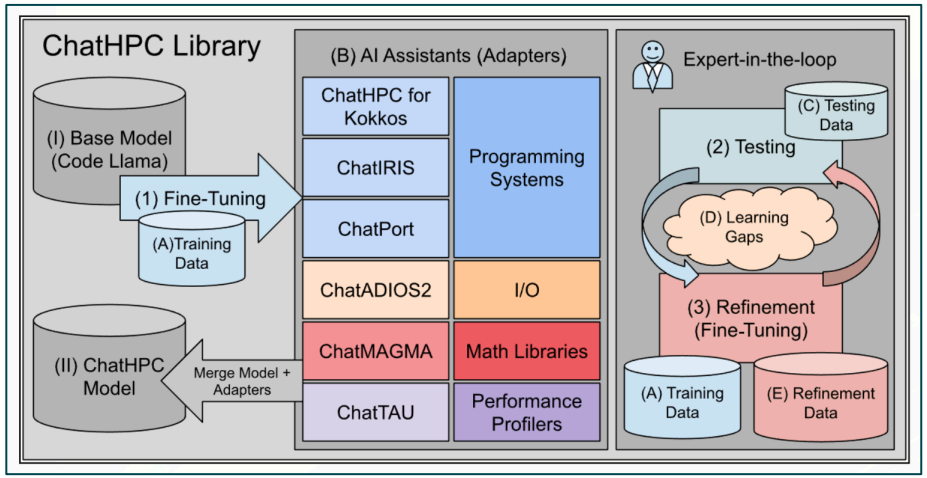


# Design and Implementation

A simple 3-step iterative process:

1. Fine-tuning -> AI assistant (optimizers)
   - Training Data
   - The better the data the better the model
   - It is more about quality than quantity
2. Testing -> Learning gaps
   - Testing Data != Training Data
   - Testing Data may contain question about non-fined-tuned capabilities
3. Refinement -> More capable AI assistants
   - Expert-Supervised Refinement Data

AI assistants can be merge into a single LLM: ChatHPC model




```
[
 {"question": "Can you translate this HIP code to Kokkos?
  __global__ void stencil (int n, float dx, float dy, float *u,
      float *u_new, float *f) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if(i > 0 && i < n-1 && j > 0 && j < n-1){
     u_new[i][j] = 0.25 * (u[i-1][j] + u[i][j+1]
                 + u[i][j-1] + u[i+1][j] + f[i][j]);
    }
  }",
  "context": "HIP to Kokkos translation",
  "answer":
  "Kokkos::parallel_for(\"stencil\",
     Kokkos::MDRangePolicy<Kokkos::Rank<2>>
              ({1, 1}, {n-1, n-1}),
     KOKKOS_LAMBDA(const int i, const int j) {
      u_new(i,j) = 0.25 * (u(i-1,j) + u(i,j+1)
                 + u(i,j-1 + u(i+1,j) + f(i,j));
  });"
  },
  ...
]
```

# Analysis

- Code Llama 7B [13GB]
  - ChatGPT 4o 1.8T -> 257x less parameters
- AI assistants (~100MB)
- Training data (~KB)
- Evaluation data (~KB)
- Fine-tuning:
  - No more than 15 min. in a one NVIDIA 2x H100 node
- Evaluation
  - % of prompts (evaluation data) passed
  - Evaluation data is different to training data
- Compare against
  - 7 billion parameter Meta's Code Llama base model (no fine-tuning)
  - 1.8 trillion parameter OpenAI's ChatGPT 4o model

- Data
  - Synthetic data in most of the AI-assistants created (supervised learning)
    - Kernels
    - Applications
    - Scripts
  - Applications:
    - Dense/Sparse Linear Algebra, PDEs, CFD, AI, Image Processing, …
  - Expert-in-the-loop
    - If we want AI responding as expert, we need data provided by experts
    - Create data
    - Identify learning gaps
  - Minimize data demanding
  - Self-learning
  - Reinforcements, if necessary (learning gaps)
  - Which targets are more demanding?
  - Not use all data (training data) we can use

# ChatHPC for Kokkos

| Model/Context | CodeLlama | ChatHPC for Kokkos Initial | ChatHPC for Kokkos Refinement | ChatGPT |
|---|---|---|---|---|
| **Documentation** | 9.5% | **89.0%** | - | 81.0% |
| **Installation** | 27.2% | **78.0%** | - | 45.5% |
| **Development** | 0.0% | **85.0%** | - | 15.1% |
| **Parallelization** | 0.0% | 45.5% | **90.9%** | 66.7% |
| **Portability** | 0.0% | 55.6% | **85.9%** | 33.4% |
| **CUDA*** | 0.0% | 53.2% | **87.2%** | 41.3% |
| **OpenACC*** | 0.0% | 58.7% | **83.5%** | 31.2% |

Kokkos: A C++ API portable library for parallel programming

Parallelization and Portability

- Training (HPC and scientific kernels)
  - BLAS, Sparse, Euler methods, Conjugate Gradient, LBM, AI convolutions
- Learning gaps
  - Sparse Computation and Iterative Solvers
  - 12% more data

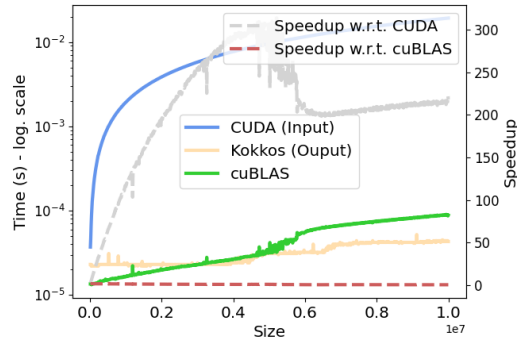
No CUDA and OpenACC codes used for fine-tuning

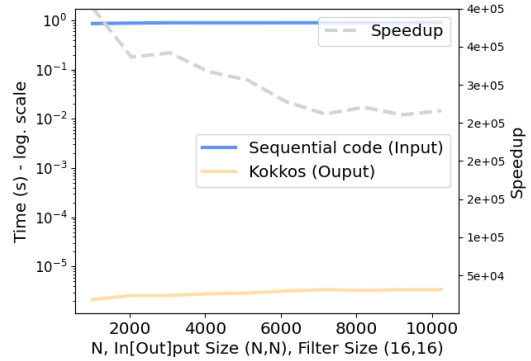
Translate the next CUDA code to Kokkos:

```
__global__ void sdot(int n, float *x, float *y, float *result) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  float update = 0.0;
  if (i < SIZE) {
    update = x[i] * y[i];
    atomicAdd(result, update);
  }
}
```

*ChatKokkos* output:

```
Kokkos::parallel_reduce(SIZE,
  KOKKOS_LAMBDA(int i, float &update){
    update += x(m) * y(m);
  },
  Kokkos::Sum<float>(result));
```



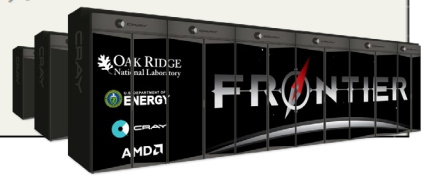



# ChatADIOS2

| Model/Context | CodeLlama | ChatADIOS2 Intial | ChatADIOS2 Refinement | ChatGPT |
|---|---|---|---|---|
| **Introduction** | 0.0% | 77.0% | **100.0%** | 33.3% |
| **Variable Definition** | 0.0% | 50.0% | **100.0%** | 33.3% |
| **Data Compression** | 0.0% | 25.0% | **100.0%** | 0.0% |
| **Parallelization** | 0.0% | 0.0% | **100.0%** | **100.0%** |

```
adios2::ADIOS adios("adios2_config.yaml", mpiComm);
adios2::IO io = adios.DeclareIO("WriteIO");
adios2::Variable<double> varU =
      io.DefineVariable<double>("U",shape,start,count);
adios2::Operator Zfp =
      adios.DefineOperator("Zfp", adios2::ops::LossyZFP);
varU.AddOperation(Zfp,{{adios2::ops::zfp::key::rate,"9"}});

adios2::Engine bpWriter =
      io.Open("output.bp", adios2::Mode::Write);
while( run_simulation ){
      bpWriter.BeginStep();
      bpWriter.Put<double>(varU, U.data());
      ...
      bpWriter.EndStep();
}
bpWriter.Close();
```



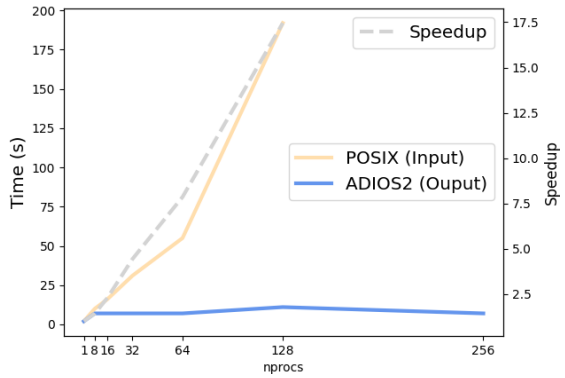

ADIOS2: A portable and HPC I/O library

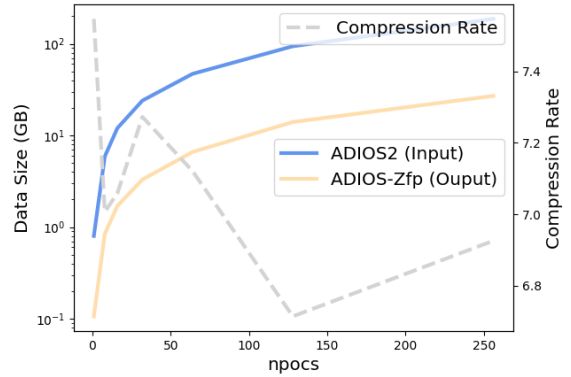
Capabilities:

- Q&A on API, Variable definition, Compression, and Parallelization (serial POSIX I/O to ADIOS2)

Learning gaps:

- Wrong use of ADIOS2 API
  - Local and global N-dimension variables
  - Compression factor (parameters)



# ChatIRIS

| Model/Context | CodeLlama | ChatIRIS Intial | ChatGPT |
|---|---|---|---|
| **Documentation** | 0.0% | **95.0%** | 70.0% |
| **OpenMP -> IRIS** | 0.0% | **82.5%** | 10.0% |
| **CUDA -> IRIS** | 0.0% | **90.0%** | 50.0% |
| **HIP -> IRIS*** | 0.0% | **81.6%** | 10.0% |

IRIS: A Task-based programming model for heterogeneous computing

- Taskification of applications
  - Deploy applications on any architecture using multiple devices
- Portability
  - OpenMP, CUDA, HIP to IRIS task code
- Training
  - Basic applications (AXPY)
- Testing
  - More complex applications with multiple kernel invocations
- No refinements were needed
- HIP was not used for fine-tuning

Translate the next CUDA code to IRIS:

```
alpha = 2.0;
X = (float *) malloc(SIZE * sizeof(float));
Y = (float *) malloc(SIZE * sizeof(float));
Z = (float *) malloc(SIZE * sizeof(float));
for (int i = 0; i < SIZE; i++) { X[i] = i; Y[i] = i;}
cudaMalloc((void**)&d_x, SIZE * sizeof(float));
cudaMalloc((void**)&d_y, SIZE * sizeof(float));
cudaMalloc((void**)&d_z, SIZE * sizeof(float));
// Initialization of arrays
cudaMemcpy(d_x, x, SIZE* sizeof(float), cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, SIZE* sizeof(float), cudaMemcpyHostToDevice);
int threads = 512;
int blocks = floor(SIZE/threads) + 1;
saxpy<<<blocks, threads>>>(alpha, d_x, d_y, SIZE);
cudaDeviceSynchronize()
sdot<<<blocks, threads>>>(d_z, d_x, SIZE);
cudaDeviceSynchronize()
cudaMemcpy(z, d_z, SIZE * sizeof(float), cudaMemcpyDeviceToHost);
cudaFree(d_x);
cudaFree(d_y);
cudaFree(d_z);
free(X);
free(Y);
free(Z);
```

*ChatIRIS* output:

```
// Architecture-agnostic IRIS task code
alpha = 2.0;
iris_init(&argc, &argv, 1);
iris_mem iris_x;
x = (float *) malloc(SIZE * sizeof(float));
iris_data_mem_create(&iris_x, x, SIZE * sizeof(float));
iris_mem iris_y;
y = (float *) malloc(SIZE * sizeof(float));
iris_data_mem_create(&mem_Y, Y, SIZE * sizeof(float));
iris_mem iris_z;
z = (float *) malloc(SIZE * sizeof(float));
iris_data_mem_create(&mem_z, z, SIZE * sizeof(float));
// Initialization of arrays
iris_graph graph;
iris_graph_create(&graph);
iris_task saxpy_task;
iris_task sdot_task;
iris_task_create(&saxpy_task);
iris_task_create(&sdot_task);
void* saxpy_params[3]={&alpha, &iris_x, &iris_y};
int saxpy_params_info[3]={sizeof(A), iris_w, iris_r};
iris_task_kernel(task0, "saxpy", 1, NULL, &SIZE, NULL, 3,
      saxpy_params, saxpy_params_info);
void* sdot_params[2]={&iris_z, &iris_x};
int sdot_params_info[2]={iris_w, iris_r};
iris_task_kernel(task1, "sdot", 1, NULL, &SIZE, NULL, 2,
      sdot_params, sdot_params_info);
iris_graph_task(graph, saxpy_task, iris_default, NULL);
iris_graph_task(graph, sdot_task, iris_default, NULL);
iris_graph_submit(graph,iris_any,1);
free(X);
free(Y);
free(Z);
iris_finalize();
```

# ChatMAGMA

| Model/Context | CodeLlama | ChatMAGMA Initial | ChatMAGMA Refinement | ChatGPT |
|---|---|---|---|---|
| **MKL -> MAGMA** | 0.0% | 37.5% | **93.0%** | 0.0% |
| **cuBLAS/Solver -> MAGMA** | 0.0% | 50.0% | **96.5%** | 0.0% |
| **hipBLAS/Solver -> MAGMA** | 0.0% | 45.3% | **95.3%** | 0.0% |

MAGMA: A portable (CPU+GPU) BLAS and LAPACK library with +400 routines

Capabilities:

- Porting of vendor (arch)-specific applications to MAGMA
  - Support for heterogeneous (CPU+ GPU) computing
- Training
  - Well known BLAS (dgemm, dtrsm) and LAPACK (dgetsv, dpotrf) for double precision
- Testing
  - Rest of BLAS and LAPACK for other precisions
- Learning gaps:
  - MAGMA-specific data types (queues) for different precisions
  - 51.72% extra data

No AMD math libraries codes were used for fine-tuning

Translate the next Intel MKL code to MAGMA:

```
// Intel MKL (CPU-only) dgetrf code
double *A=(double *)mkl_malloc(n*n*sizeof(double),64);
int *ipiv=(int *)mkl_malloc(n*sizeof(int),64);
LAPACKE_dgetrf(LAPACK_ROW_MAJOR,n,n,A,n,ipiv);
mkl_free(A);
mkl_free(ipiv);
```

Translate the next NVIDIA cuSolver code to MAGMA:

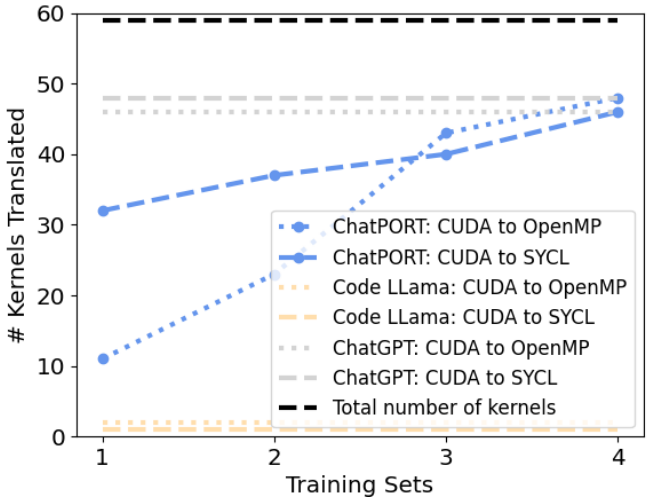
```
// NVIDIA cuSolver (GPU-only) dgetrf code
int lwork=0;
double *dwork;
int *dipiv;
double *dinfo;
cusolverDnHandle_t handle;
cusolverDnCreate(&handle);
double *hA=(double *)malloc(n*n*sizeof(double));
cusolverDnDgetrf_bufferSize(handle,n,n,dA,n,&lwork));
cudaMalloc((void **)&dA,n*n*sizeof(double));
cudaMalloc((void **)&dwork,sizeof(double)*lwork);
cudaMalloc((void **)&dipiv, n*sizeof(int));
cudaMalloc((void **)&dinfo,sizeof(int));
cudaMemcpy(dA,hA,n*n*sizeof(double),cudaMemcpyHostToDevice);
cusolverDnDgetrf(handle,n,dA,n,dwork,dipiv,dinfo);
cudaMemcpy(hA,dA,n*n*sizeof(double),cudaMemcpyDeviceToHost);
cudaFree(dA);
cudaFree(dwork);
cudaFree(dipiv);
cudaFree(dinfo);
free(hA);
cusolverDnDestroy(handle);
```

*ChatMAGMA* output:

```
// MAGMA (CPU+GPU) dgetrf code
magma_queue_t queue;
magma_device_t cdev;
magma_getdevice(&cdev);
magma_queue_create(cdev,&queue );
double *hA;
magmaDouble_ptr dA;
magma_int_t *ipiv;
magma_int_t info;
magma_dmalloc_cpu(&hA,n*n);
magma_dmalloc(&dA,n*n);
magma_imalloc_cpu(&ipiv,n);
magma_dsetmatrix(n,n,hA,n,dA,n,queue);
magma_dgetrf_gpu(n,n,dA,n,ipiv,&info);
magma_dgetmatrix(n,n,dA,n,hA,n,queue);
magma_free(dA);
magma_free_cpu(hA);
magma_free_cpu(ipiv);
magma_queue_destroy(queue);
```

# ChatPORT(ing)

| Training Set | #Kernels | Kernels |
|---|---|---|
| 1 | 5 | accuracy, geodesic, lr, maxpool3d, perplexity |
| 2 | 10 | damage, knn, heat2d, laplace3d, md |
| 3 | 15 | advCubatureHex3D, backprop, channelShuffle, lr, meanshift |
| 4 | 22 | chemv, clenergy, pathfinder, pointwise, swish, stddev, tissue |




- Create an AI assistant for code portability from vendor-specific models (e.g., CUDA) to SYCL and OpenMP-offloading
- Data is provided by HeCBench
  - Training:
    - 4 different data sets with (5, 10, 15, and 22) kernels from multiple different domains
  - Testing
    - 59 CUDA kernels to be ported to OpenMP-offloading and SYCL

# Remarks

| Assistant | Code Llama | Initial | Extra Data | Refinement | ChatGPT |
|---|---|---|---|---|---|
| **ChatHPC for Kokkos** | 7.34% | 70.51% | 12.33% | **85.75** | 48.34% |
| **ChatADIOS2** | 0.00% | 40.87% | 37.50% | **100.00%** | 25.0% |
| **ChatMAGMA** | 0.00% | 44.26% | 51.72% | **94.93%** | 0.00% |
| **ChatIRIS** | 0.00% | **87.29%** | - | - | 35.00% |
| **ChatPort** | 2.54% | 36.44% | 77.27% | **79.66%** | **79.66%** |

- New HPC-specific capabilities on top of Code Llama across different HPC domains
  - Programming models, I/O, math libraries, tooling, and runtimes

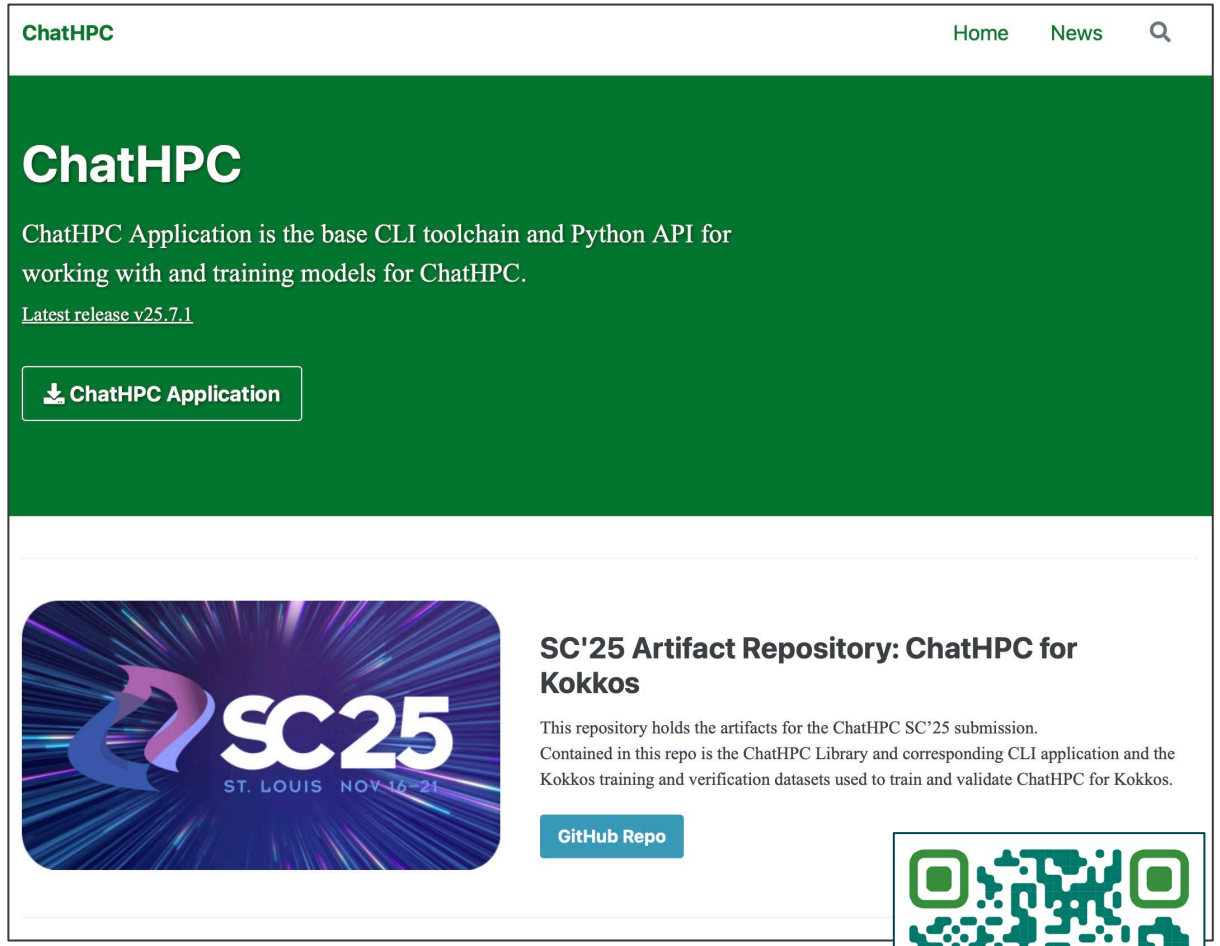
  And tasks:
  - Parallelization, portability, performance analysis, scalability, etc.
- Provide higher levels (of up to 90% higher) of trustworthiness than ChatGPT quickly with modest resources
  - Increase data-size (expert-in-the-loop) during refinement learning has a higher impact on correctness
- More data (prompts) for testing than for training data
  - Testing data contains prompts about non-fine-tuned capabilities

# Future Work

- **ChatMPI**
  - A ChatHPC AI-assistant for MPI parallelization
    - 4x speedup w.r.t. MPI code generated by ChatGPT-5
- **3 Pillars**
  - **Multimodality**
    - Simplify the interaction with AI for HPC domains
  - **Reasoning**
    - Generate highly optimized HPC codes across domains and technologies
  - **Agentic**
    - Connect HPC and AI for complex problems

# Resources and Community

- ChatHPC project:
  https://ornl.github.io/ChatHPC/
- Meetings:
  - We organize a community meeting once a month.
  - Join us!
- Workshops:
  - LLM4HPC (co-located with ISC-HPC):
    https://ornl.github.io/events/llm4hpc2025/
  - LLM4HPCAsia (collocated with SCA/HPC Asia)
    https://ornl.github.io/events/llm4hpcasia2026/



# Observations (Things to think about)

- **Fine-tuning**
  - Very effective, if you have the data
- **Trustworthiness**
  - You can elevate trustworthiness (BTW: better than ChatGPT)
- **Overfitting**
  - Is that a problem for domain-specific AI assistants?
- **Self-learning**
  - Train on simple problems and let LLMs to deal with more complex problems
  - Take advantage from similarities (similar patterns)
- **Synthetic or real data?**
  - Why not both?
- **Expert-in-the-loop**
  - Recommendable for domain-specific LLMs
- **Accessibility**
  - One NVIDIA GPU (better if you have 2), data (~KB), and 15 minutes
- **Simplify prompting**
  - No prompt engineering, more productive, non-experts can generate expert level responses
- **Impact on HPC**
  - Created capabilities for programming models, I/O, math libraries, performance tools, …
  - **Help our community to use the HPC tools we have today**
- **Leveraging HPC efforts**
  - ChatHPC benefits from the big effort made during last years

# Use ChatHPC!

OAK RIDGE National Laboratory

SC25 ST. LOUIS NOV 16–21

# ChatHPC:
## Building the foundation of an AI assisted and productive HPC ecosystem

# Thanks!


**PRESENTED BY**

Pedro Valero-Lara

PhD. Sr Computer Scientist <valerolarap@ornl.gov>


U.S. DEPARTMENT OF ENERGY

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY


**Acknowledgments:**

This research used resources of the Oak Ridge Leadership Computing Facility and the Experimental Computing Laboratory at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract No. DE-AC05-00OR22725. This material is based upon work supported by the US Department of Energy, Office of Science's Advanced Scientific Computing Research program as part of the Advancements in Artificial Intelligence for Science program, Ellora and Durban projects, the Next Generation of Scientific Software Technologies program, S4PST and PESO projects, and the Scientific Discovery through Advanced Computing (SciDAC) program, RAPIDS2 SciDAC Institute for Computer Science, Data, and Artificial Intelligence.