OAK RIDGE National Laboratory

July 2026 | Scalable Tools Workshop 2026

GenASPEN: an agentic AI ecosystem for performance modeling and prediction

Grace McLewee, University of Oregon

Mentor: Mohammad Alaul Haque Monil, ORNL

Advisor: Allen Malony, UO

Collaborators: Pedro Valero-Lara, William Godoy, Keita Teranishi, Narasinga Rao, Jeffery Vetter

Architectures and Performance Group

U.S. DEPARTMENT OF ENERGY

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

Introduction

What Problem Are We Solving?

- **Performance model tools (ASPEN) require human effort (COMPASS)**
- **LLMs strengths in language, pattern recognition, but lack determinism**
- **Agentic AI Solution** – Combine LLM strengths and accurate, deterministic performance modeling tools

Can an agentic workflow deliver accurate prediction with no compiler-level effort?

Objective: Automate ASPEN performance modeling with LLMs

Aspen:

- A domain-specific language (DSL) for analytical performance modeling
- Provides two formal model types:
    - **Application model** — parallelism, operation counts, data structures, control flow
    - **Abstract machine model** — hardware characteristics (nodes, sockets, cores, memory, interconnect)
- Models are compiled, checked for correctness, then queried/simulated for performance predictions
- Key advantages over ad-hoc analytical modeling:
    - Enforces consistency and correctness through formal grammar
    - Enables modular, reusable, composable models
    - Positioned between structured analytical models and functional simulation — fast and flexible, but less detailed than full simulation

K. L. Spafford and J. S. Vetter, "Aspen: A domain specific language for performance modeling," SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 2012, pp. 1-11, doi: 10.1109/SC.2012.20.

ASPEN DSL For Performance Modeling

Listing 1: Input OpenACC Matrix Multiplication Code

```
1  int N = 1024;
2  void matmul(float *a, float *b, float *c){ int i, j, k ;
3  #pragma acc kernels loop gang copyout(a[0:(N*N)]) \
4  copyin(b[0:(N*N)],c[0:(N*N)])
5    for (i=0; i<N; i++){
6  #pragma acc loop worker
7      for (j=0; j<N; j++) { float sum = 0.0 ;
8        for (k=0; k<N; k++) {sum+=b[i*N+k]*c[k*N+j];}
9        a[i*N+j] = sum; }
10   } //end of i loop
11 } //end of matmul()
12 int main() {
13   int i; float *A = (float*) malloc(N*N*sizeof(float));
14   float *B = (float*) malloc(N*N*sizeof(float));
15   float *C = (float*) malloc(N*N*sizeof(float));
16   for (i = 0; i <  N*N; i++)
17   {  A[i] = 0.0F; B[i] = (float) i; C[i] = 1.0F; }
18 #pragma aspen modelregion label(MM)
19   matmul(A,B,C);
20   free(A); free(B); free(C); return 0;
21 } //end of main()
```

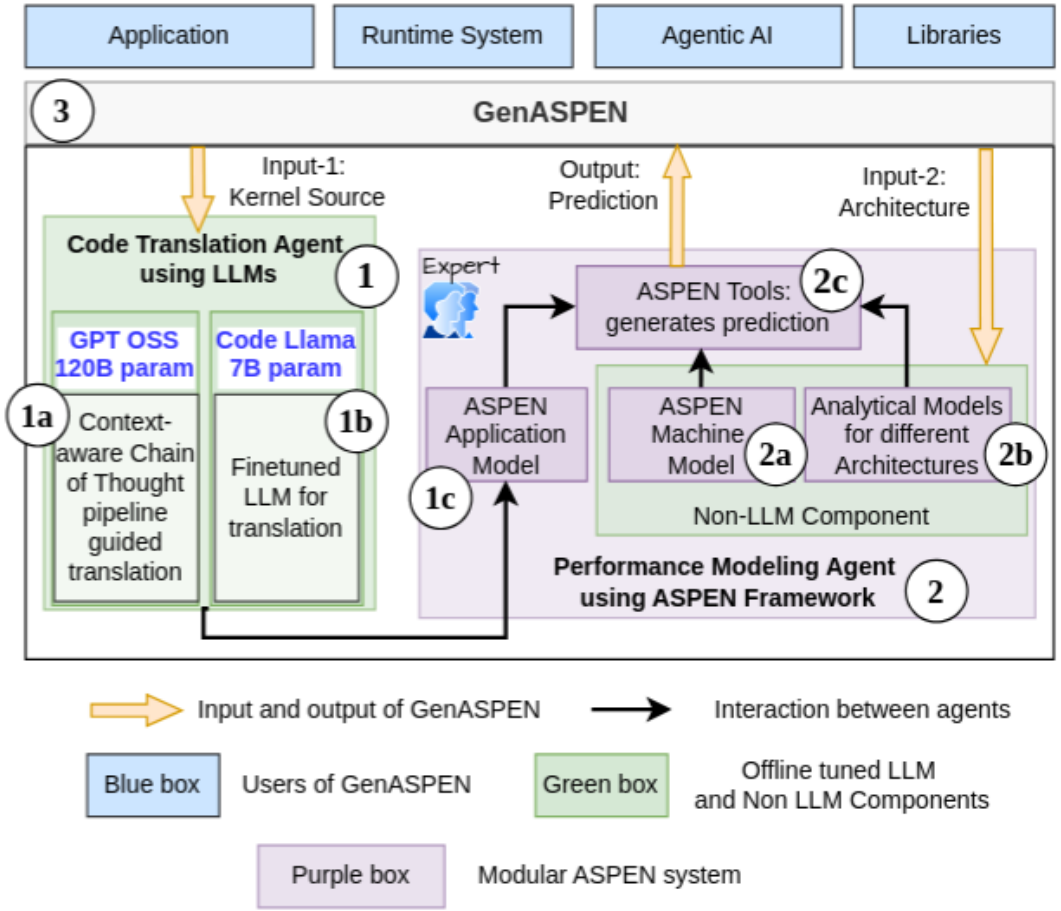
COMPASS annotations and automatically generating ASPEN application models

Listing 2: Output Aspen Performance Model. Note that Aspen *param* variables are parameterizable; the Aspen modeling framework can change their values as necessary.

```
1  model MM {
2    param floatS = 4; param N = 1024
3    data A as Array((N*N), floatS)
4    data B as Array((N*N), floatS)
5    data C as Array((N*N), floatS)
6    kernel matmul {
7      execute matmul2_intracommIN
8      {  intracomm [floatS*(N*N)] to C as copyin
9        intracomm [floatS*(N*N)] to B as copyin }
10     map matmul2 [N] {
11       map matmul3 [N] {
12         iterate [N] {
13           execute matmul5
14           {  loads [floatS] from B as stride(1)
15             loads [floatS] from C; flops [2] as sp, simd }
16         } //end of iterate
17         execute matmul6 { stores [floatS] to A as stride(1) }
18       } // end of map matmul3
19     } //end of map matmul2
20     execute matmul2_intracommOUT
21     { intracomm [floatS*(N*N)] to A as copyout }
22   } //end of kernel matmul
23   kernel main { matmul() }
24 } //end of model MM
```


Lee, Seyong, Jeremy S. Meredith, and Jeffrey S. Vetter. "Compass: A framework for automated performance modeling and prediction." *Proceedings of the 29th ACM on International Conference on Supercomputing*. 2015.

GenASPEN Workflow



The workflow (3) uses Langgraph

Steps:

LLM parses natural language input

Kernel code and user input parameters

Target device

Prediction (runtime, loads, stores, flops)

2. LLM (1) invokes translation method (1a or 2b)

3. LLM (2) calls ASPEN tools (2c) to traverse ASPEN application model (1c) and ASPEN Machine model (2a)

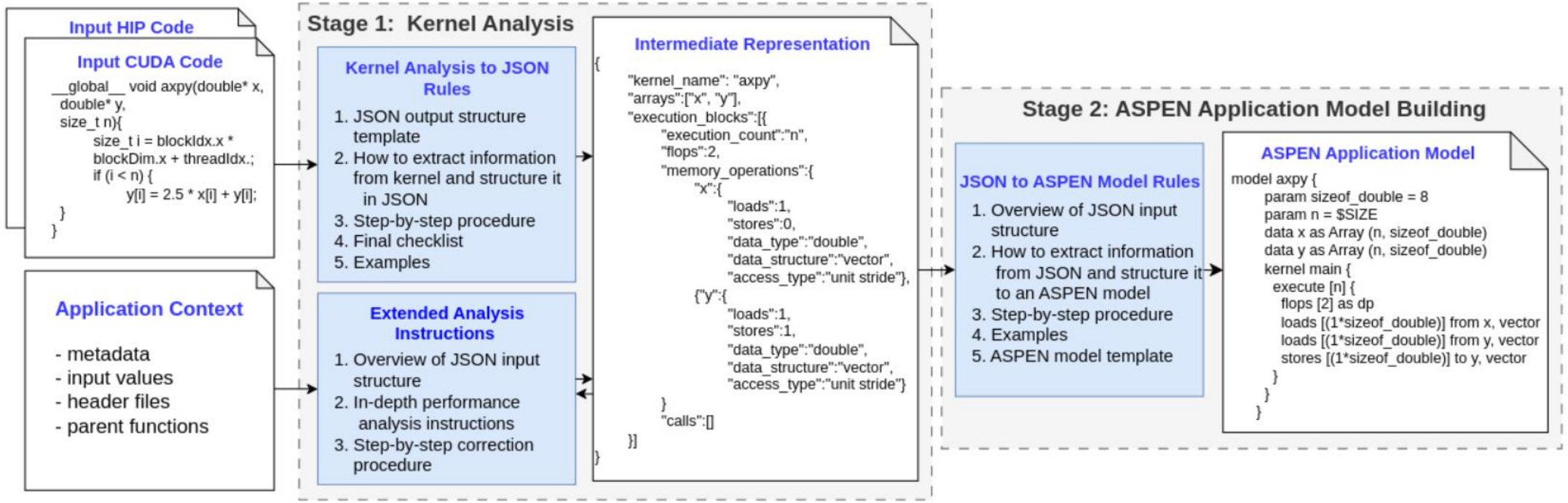
Analytical models stored in (2a)

4. LLM (2) receives tool output and returns

Fine-Tune to Reasoning

	Fine Tuned LLM (ChatHPC)	Prompt Engineering
Model	Code Llama 7B	GPT OSS 120B
Prompt	Zero shot	Context Aware, Chain of Thought (CA-CoT)
Pros	Fast inference	Generalization
Cons	Data scarcity Memory constraints Overfitting	Context Window Challenges Prompt size Needle in haystack Larger token requirement

CA-CoT: Analyze First, Synthesize Second



Stage 1: Teach LLM how to do performance analysis

- How to count operations, identify memory access patterns, etc.

Stage 2: Teach LLM how to build an ASPEN application model

- What the JSON IR means
- ASPEN grammar rules
- How to structure ASPEN application model

ASPEN Application Model Generation Trade-Offs

TABLE III: Application model generation accuracy of GenASPEN using GPT-OSS (120 billion parameters) and Code Llama models (7 billion parameters) for various HPC benchmarks and applications. Here, Speedup = [inference time of CA-CoT] / [inference time of fine-tuning] and Token Usage % = [token count of fine-tuning]/[token count of CA-CoT]*100.

Benchmarks	Bound by	Memory Access	Complexity	Syntactical Accuracy [Inference Time (s)] [Token Count]		Fine-Tuning vs. CA-CoT	
				Fine-tuning	CA-CoT	Speedup	Token Usage (%)
BLAS Level 1	Memory	Regular	Low	Correct [0.80] [131]	Correct [55.32] [2.7K]	69×	4.85%
BLAS Level 2	Memory	Regular	Moderate	Correct [0.92] [150]	Correct [58.88] [3.1K]	64×	4.84%
BLAS Level 3	Compute	Regular	Moderate	Correct [0.98] [162]	Correct [65.71] [3.5K]	67×	4.64%
Deep Learning	Memory	Regular	Low	Correct [0.72] [115]	Correct [72.5] [4.2K]	101×	2.74%
PDEs	Memory	Regular	Moderate	Correct [1.56] [132]	Correct [107.34] [7.6K]	69×	1.74%
Sparse	Hybrid	Irregular	High	Correct [16.62] [234]	Correct [37.58] [11K]	2.3×	2.13%
CG	Memory	Regular	High	Correct [4.43] [235]	Correct [41.00] [12.5K]	9.3×	1.88%
XSBench	Memory	Irregular	Very High	**Incorrect** [8.06] [1K]	Correct [344.25] [37.4K]	43×	2.67%
MiniMDock	Hybrid	Regular	Very High	**Incorrect** [338.02] [59.3K]	Correct [420.88] [152.2K]	1.2×	38.96%
LULESH	Memory	Irregular	Very High	**Incorrect** [248.58] [8.3K]	Correct [1023.18] [393.7K]	4.1×	2.11%

Takeaways:

- Fine-tuned LLM: generates accurate ASPEN application models similar to those in the training set, fails otherwise, fast inference time and low token cost
- CA-COT: Generates accurate ASPEN models for in all cases, but for longer time and higher token cost

(time cost largely due to slow Ollama server)

Pattern-Specific Analytical Model

the interface ASPEN provides

The ASPEN application model hands the tool four things:

- FLOPs
- loads
- stores
- an access descriptor (added new ones to grammar)

Documentation lost...

obj: find anything that works empirically

- λ per term = measured actual ÷ base prediction, against a unit-stride global baseline
- Stencil and sparse: λ on loads only, as a function of stencil points / nnz
- Up to 12 points per pattern, fit per architecture

$$T_{\text{exec}} = \max(T_{\text{compute}}, T_{\text{memory}}) \quad (1)$$

The compute time is given by:

$$T_{\text{compute}} = N \cdot \frac{F_{\text{ops}}}{P\_FLOPS_a \cdot \lambda} \quad (2)$$

The memory time varies by access pattern:

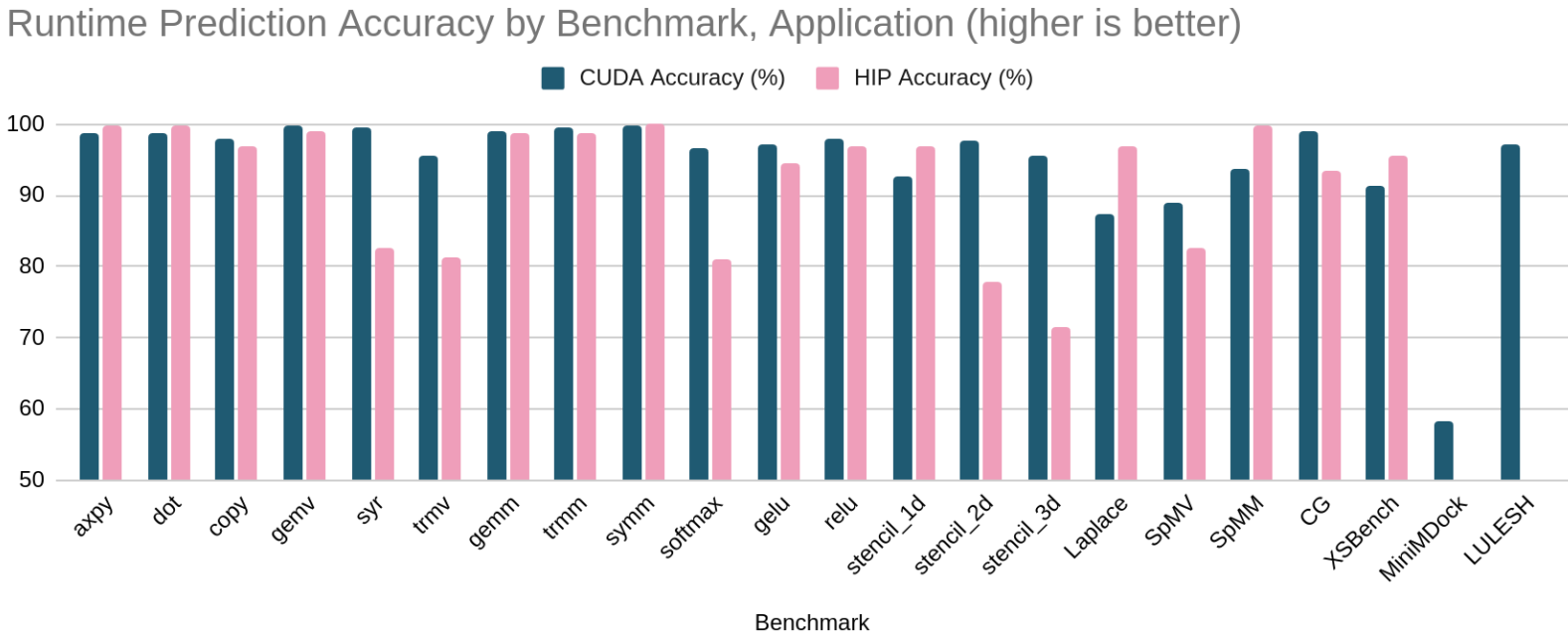
$$T_{\text{pattern}} = (L + S) \cdot N \cdot \frac{\text{sizeof(data\_type)}}{BW_a} \cdot \lambda_{\text{pattern}} \quad (3)$$

pattern $\in$ {stride(n), matrix-vector, matrix-matrix, stencil_1d, stencil_2d, stencil_3d, sparse, shared memory}, $n \in Z^+$.

Op Counts and Runtime

Kernel	FLOPS %	MOPS %
Benchmarks, CG	100	100
XSBench	94.53	90
MiniMDock	86.4	82.95
LULESH	86.14	91.39

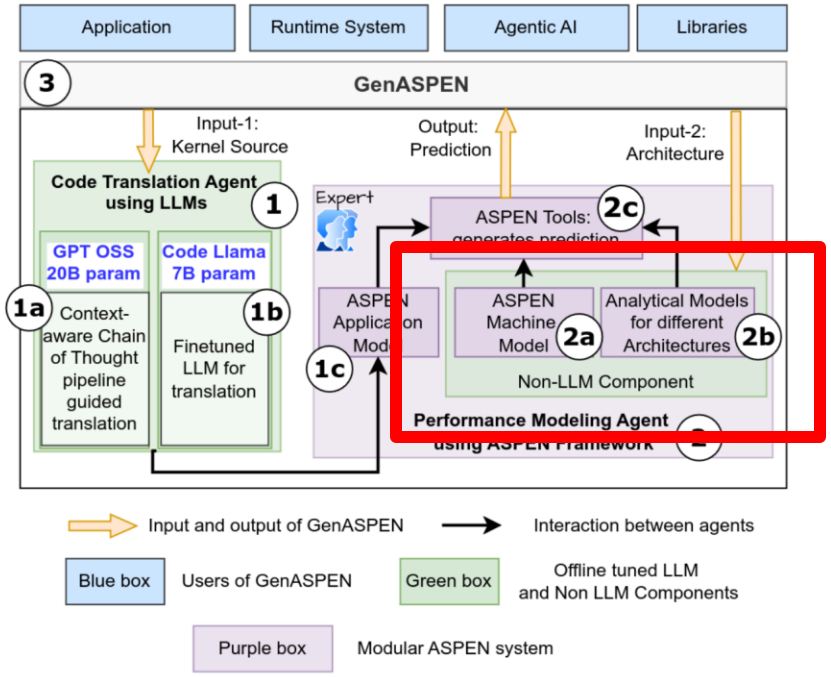
- GPT OSS 120B unable to reason about runtime parameters
- Passed dynamic information via user prompt
- Challenge in structuring prompts and tooling for proxy application

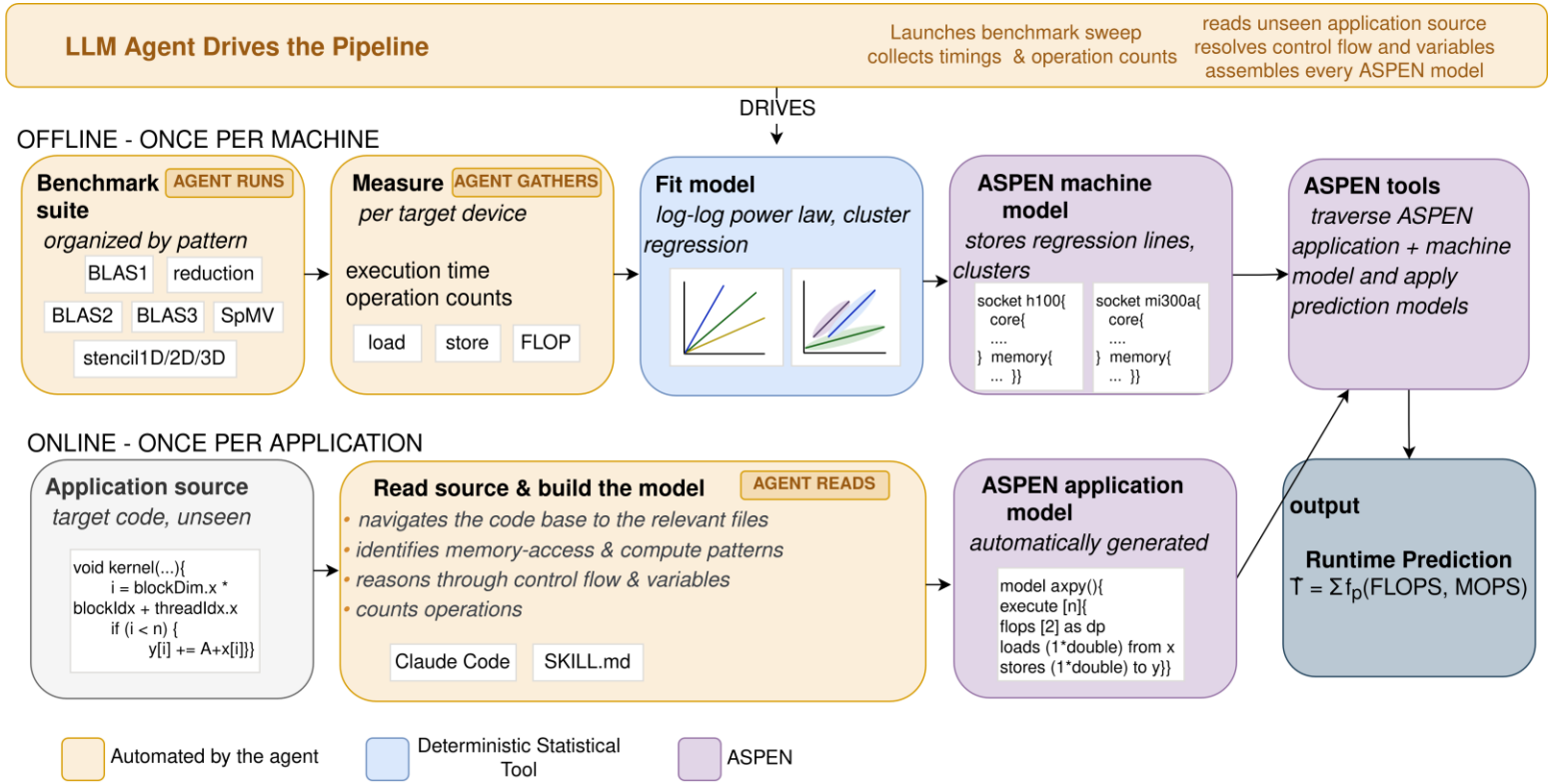


stencil_3d HIP (71.44), stencil_2d HIP (77.95), MiniMDock (58.34)

HIP ommited for LULESH, MiniMDock (only CUDA versions available)

Extending GenASPEN





Goal: Use LLMs to automate "Non-LLM Component" for fully automatic performance modeling
In-place performance modeling for large applications

Conclusion

Outcome: LLMs able to

1. translate source code to ASPEN DSL
2. count operations
3. identify memory access patterns

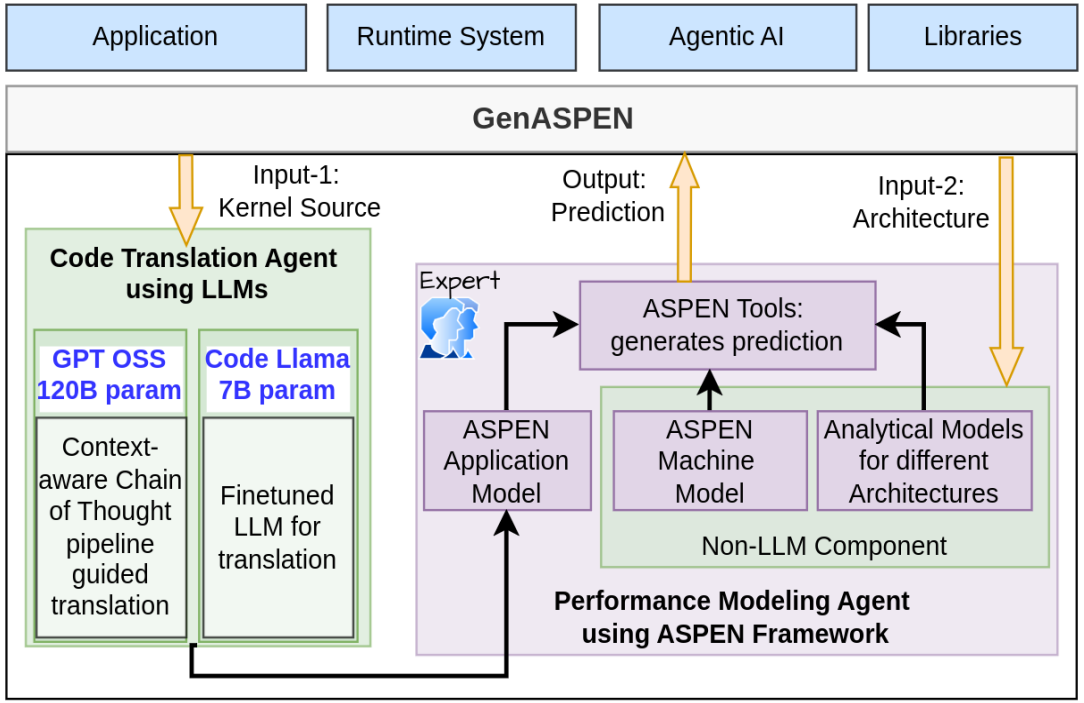
Tradeoffs:

- Fine-tune: Small kernels, quick translation
- Reasoning: Complex codes, higher token usage

Results: execution time predictions 94.64% accuracy on NVIDIA and 92.19% accuracy on AMD GPUs

Future work: Add tools for more robust prediction models

This work has been been supported by DOE ASCR AI for Science projects, Durban and Ellora.



References

G. McLewee, M. A. H. Monil, P. Valero-Lara, N. Miniskar, W. Godoy, A. Malony, K. Teranishi, and J. S. Vetter, “GenASPEN: An agentic AI framework for automatic performance modeling and prediction of HPC codes,” in Proc. 2026 IEEE Int. Conf. Cluster Comput. (CLUSTER), Alexandria, VA, USA, 2026, accepted for publication.

Lee, Seyong, Jeremy S. Meredith, and Jeffrey S. Vetter. "Compass: A framework for automated performance modeling and prediction." *Proceedings of the 29th ACM on International Conference on Supercomputing*. 2015.

Monil, Mohammad Alaul Haque, et al. "MAPredict: Static Analysis Driven Memory Access Prediction Framework for Modern CPUs." *International Conference on High Performance Computing*. Cham: Springer International Publishing, 2022.

Karlin, Ian, Jeff Keasler, and J. Robert Neely. *Lulesh 2.0 updates and changes*. No. LLNL-TR--641973. Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States), 2013.

Tramm, John R., et al. "XSBench-the development and verification of a performance abstraction for Monte Carlo reactor analysis." *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)* (2014).

Thavappiragasam, M., et al.: Performance portability of molecular docking miniapp on leadership computing platforms. In: IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC) (2020)

Supplemental Material: Application Model GenASPEN

```
model gemv{
  param sizeof_double = 8
  param n = 1
  data a as Array((n*n), sizeof_double)
  data b as Array(n, sizeof_double)
  data c as Array(n, sizeof_double)
  kernel main{
    execute [n*n] {
      flops [2] as dp
      loads [(1*sizeof_double)] from a as stride(1), matrix
      loads [(1*sizeof_double)] from b as stride(0), vector
      stores [(1*sizeof_double)] to c as stride(0), vector
    }
  }
}
```

Machine Model GenASPEN

```
memory mem {
  property capacity [cap]
  property bandwidth [bw]
  property stream_float_bandwidth [stream_float_bandwidth_val]

  property mv_xl1 [1.2] //for strided 1 load
  property mv_xs [0.01] // for strided n store
  property mv_xs1 [3] // for strided 1 store

  property mm_xl1 [1.095]
  property mm_xln [0.8]
  property mm_xs [0.08]

  property stencil_constant_1 [14.81405077]
  property stencil_constant_2 [0.1934093]
  property stencil_constant_3 [0.03234679]
  property stencil_constant_4 [-14.09288265]
  property stencil_constant_5 [0]
```

Langgraph Nodes: Benchmark Kernel

```python
if translate_mode == "CA CoT":
    graph.add_edge("parse_input", "check_existing")
    graph.add_conditional_edges(
        "check_existing",
        route_check_cacot,
        {
            "invoke_aspen_tool": "invoke_aspen_tool",
            "get_json_ir": "get_json_ir"
        }
    )
    if extended:
        graph.add_edge("get_json_ir", "get_json_ir_ext")
        graph.add_conditional_edges(
            "get_json_ir_ext",
            route_json,
            {
                "get_json_ir_ext": "get_json_ir_ext",
                "get_json_ir": "get_json_ir",
                "cacot_translation": "cacot_translation"
            }
        )
    else:
        graph.add_conditional_edges(
            "get_json_ir",
            route_json,
            {
                "get_json_ir_ext": "get_json_ir_ext",
                "get_json_ir": "get_json_ir",
                "cacot_translation": "cacot_translation"
            }
        )
    graph.add_edge("cacot_translation", "invoke_aspen_tool")
    graph.add_conditional_edges(
        "invoke_aspen_tool",
        route_expert,
        {
            "save_model": "save_model",
            "invoke_expert": "invoke_expert"
        }
    )
    graph.add_edge("invoke_expert", "invoke_aspen_tool")  # Re
    graph.add_edge("save_model", END)
```

Langgraph Nodes: large app

```python
graph = StateGraph(State)

graph.add_node("get_json_ir_large_app_kernel", get_json_ir_large_app_kernel)
graph.add_node("get_json_ir_large_app_device_function", get_json_ir_large_app_device_function)

graph.add_conditional_edges(
    START,
    route_json_ir,
    {
        "get_json_ir_large_app_kernel": "get_json_ir_large_app_kernel",
        "get_json_ir_large_app_device_function": "get_json_ir_large_app_device_function"
    }
)
graph.add_edge("get_json_ir_large_app_kernel", END)
graph.add_edge("get_json_ir_large_app_device_function", END)

app = graph.compile()
```

```python
graph = StateGraph(FinalState)

initial_state = {
    "messages": [],
    "device": target_device,
    "kernel": f"__global__ {model_name}()",
    "size": 0,
    "metric": ["runtime", "flops", "loads", "stores"],
    "application_model": aspen_models_str,
    "machine_model": None,
    "prediction": None,
    "model_valid": 0,
    "num_invoke": 0,
    "application_fn": None,
    "save": 0,
    "metadata": metadata,
    "cuda_code": "\n".join(kernels)
}

graph.add_node("resolve_params", resolve_params)
graph.add_node("invoke_aspen_tool", invoke_aspen_tool)
graph.add_node("invoke_expert", invoke_expert)
graph.add_node("check_existing", check_existing)
graph.add_node("save_model", save_model)

graph.add_edge(START, "resolve_params")
graph.add_edge("resolve_params", "invoke_aspen_tool")
graph.add_conditional_edges(
    "invoke_aspen_tool",
    route_expert,
    {
        "save_model": "save_model",
        "invoke_expert": "invoke_expert"
    }
)
graph.add_edge("invoke_expert", "invoke_aspen_tool")
graph.add_edge("save_model", END)
app = graph.compile()
```