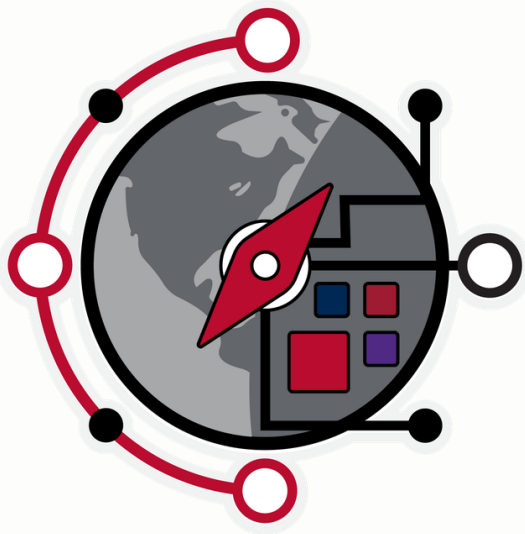




Center for
Optimized Modern
Parallel Adaptive
System Software

System Optimization: Workflow, Experiment/Data Collection ,
and Performance Tuning

PROF. TRILCE ESTRADA and PATRICK BRIDGES · UNIVERSITY OF NEW MEXICO
COMPASS 2026 ANNUAL REVIEW · OCTOBER 29, 2026 · 11:10 SESSION

Where This Work Sits on the Roadmap

	Year 1 (FY26)				Year 2 (FY27)				Year 3 (FY28)				Year 4 (FY29)				Year 5 (FY30)					
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4		
AI Performance / Modeling	Analyze AI model sensitivity				Model data collection & augmentation				Decision under uncertainty				Model testing & validation									
					Research model uncertainty quantification				Research model transfer learning				Enhanced model for optimization integration									
Cross-Layer / Optimization	Assess sparse solver optimizations		Assess Beatnik optimization		Layer-by-layer optimization		Search-based optimization		Integrate model uncertainty		Fully integrate models											
			Scheduler optimization sensitivity		Assess NNSA app optimization opportunities																	
Solver Design	Beatnik fast multipole support	Z-Model adaptive meshing																				
	Develop compressible interface model			Model topological transitions				Multi-scale reactive interface methods				Parallelize multi-scale reactive solver				Integrate models into production codes						
System Software				Parallelize compressible interface model																		
								Combine Krylov enhancements into unified solver														
System Software	MPI/Kokkos stream triggered communication			Additional MPI/Kokkos extensions				Time-based communication primitive optimization														
	Create initial RAPIDS primitives			Additional RAPIDS extensions								Cross-layer control of system software APIs										
System Software				Kokkos memory layout optimizations				Develop System Tuning APIs														
Integration & / Assessment	Design app/system workflow																					
	Design/deploy system infrastructure			Assess NNSA application optimization				Assess integrated hydro codes														
Education & / Workforce	Create bootcamp materials			Develop tradeoff analysis materials				Create AI model course materials				Offer courses with integrated materials										
	Deploy bootcamp modules			Create conference tutorial materials				Offer HOTI/SIAM/SC tutorials														

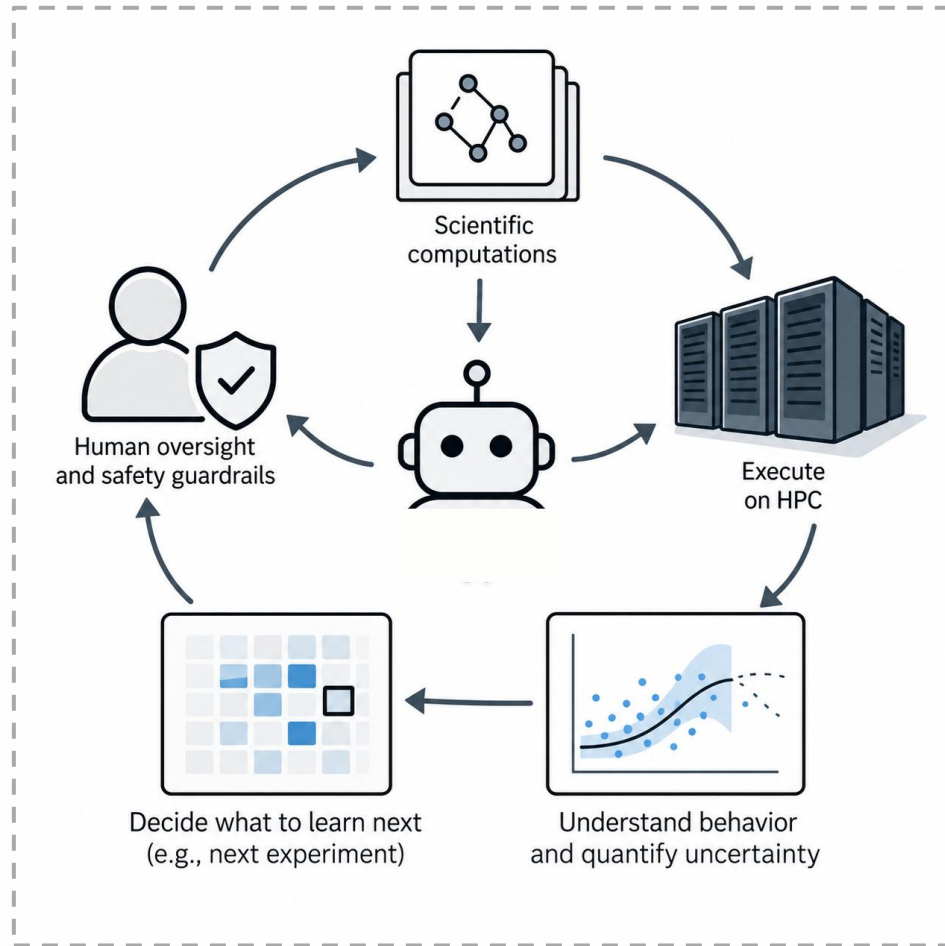
- 1 Autotuning MPI parameters**
Sensitivity, UQ, and Bayesian optimization (Barr, Kinhead)
- 2 Deciding when to deploy an optimization**
Toward layer-by-layer optimization of NNSA apps
- 3 AI tools in research**
Defaults, shared experience, Lab guidance
- 4 Next: an integrated agentic framework**
Characterization, tuning, decision, diagnosis in one loop

Toward Predictive and Adaptive Scientific Computing

Learning computational structure, modeling performance, and optimizing HPC workflows



Research Umbrella



What would it take for HPC systems to not only execute scientific computations, but also understand their behavior, recognize uncertainty, and safely guide the next experiments in an *almost* autonomous way?

Agents Plane

Interpret specifications

Generate experiments

Operate within guardrails

Adapt based on results

Application specification

Experiment Planning and Orchestration
(existing tools - optional)

Analysis and Adaptation

Multi-modal
Embedding Space

Application/System

Example (TOML)

```
description = "AMG Solver"
docs = ["docs/amg_docs.pdf"]
[run]
template = "amg -P {Px} {Py}
          {Pz} -n {nx} {ny} {nz}
          -problem {problem}"

[run.params.problem]
type = "choice"
choices = [1, 2]
description = "Problem number"
```

- Application documentation
- Parameter space and allowed values
- Resource and execution constraints

Benchmark

benchmark

Experiment definition and reproducibility

- System specifications
- Application configurations
- Spack environments
- Parameter sweeps
- Reproducible experiments

Snakemake

snakemake

Workflow construction and data pipelines

- Input generation
- Preprocessing
- Simulation workflows
- Postprocessing
- Data management

Merlin

MERLIN

Experiment orchestration and campaign management

- Parameter sweeps
- Task dependencies
- Campaign monitoring
- Fault tolerance
- Iterative execution

Flux

Flux

Resource management and scheduling

- Resource allocation
- Job scheduling
- Task placement
- Multi-node execution
- Integration with schedulers

Performance Models

- Predict performance
- Quantify uncertainty
- Identify informative experiments

Experiment results

- Runtime and scalability
- System telemetry
- Metadata and logs
- Structured dataset

Data Collection



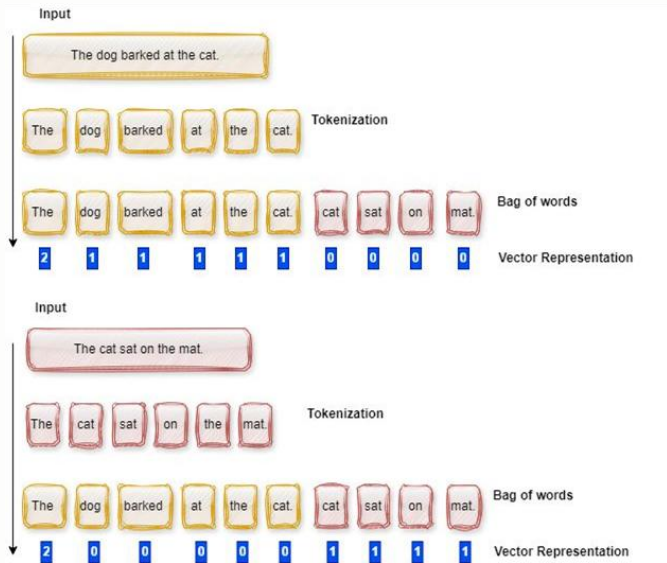
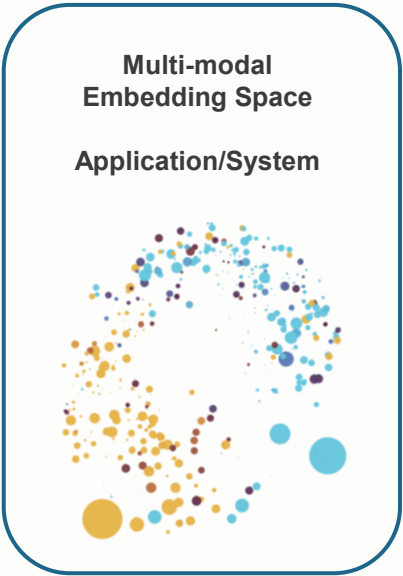
Applications



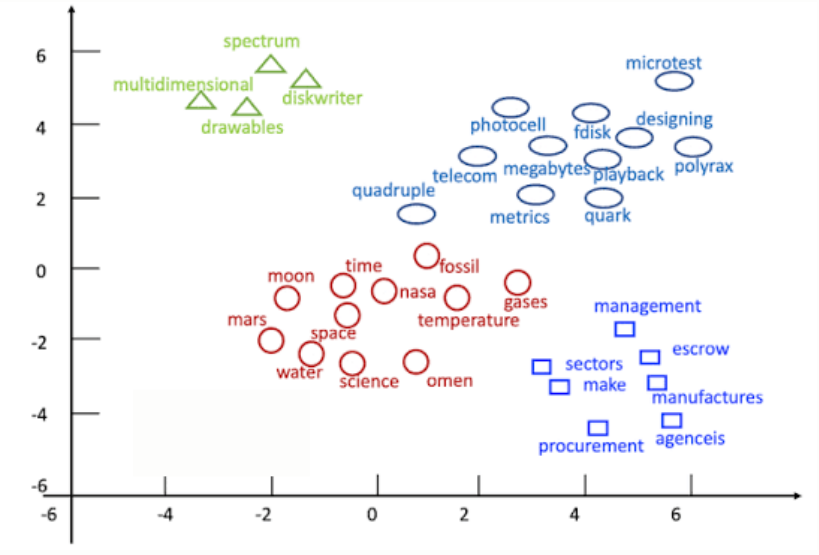
HPC Platforms

Why do we want to learn rich embedding representations?

Remember old-fashioned language modeling?



VS



**AMG**
Algebraic Multigrid

**MFEM**
Finite Element Methods

**PARTHENON**
Astrophysical Simulations

Applications

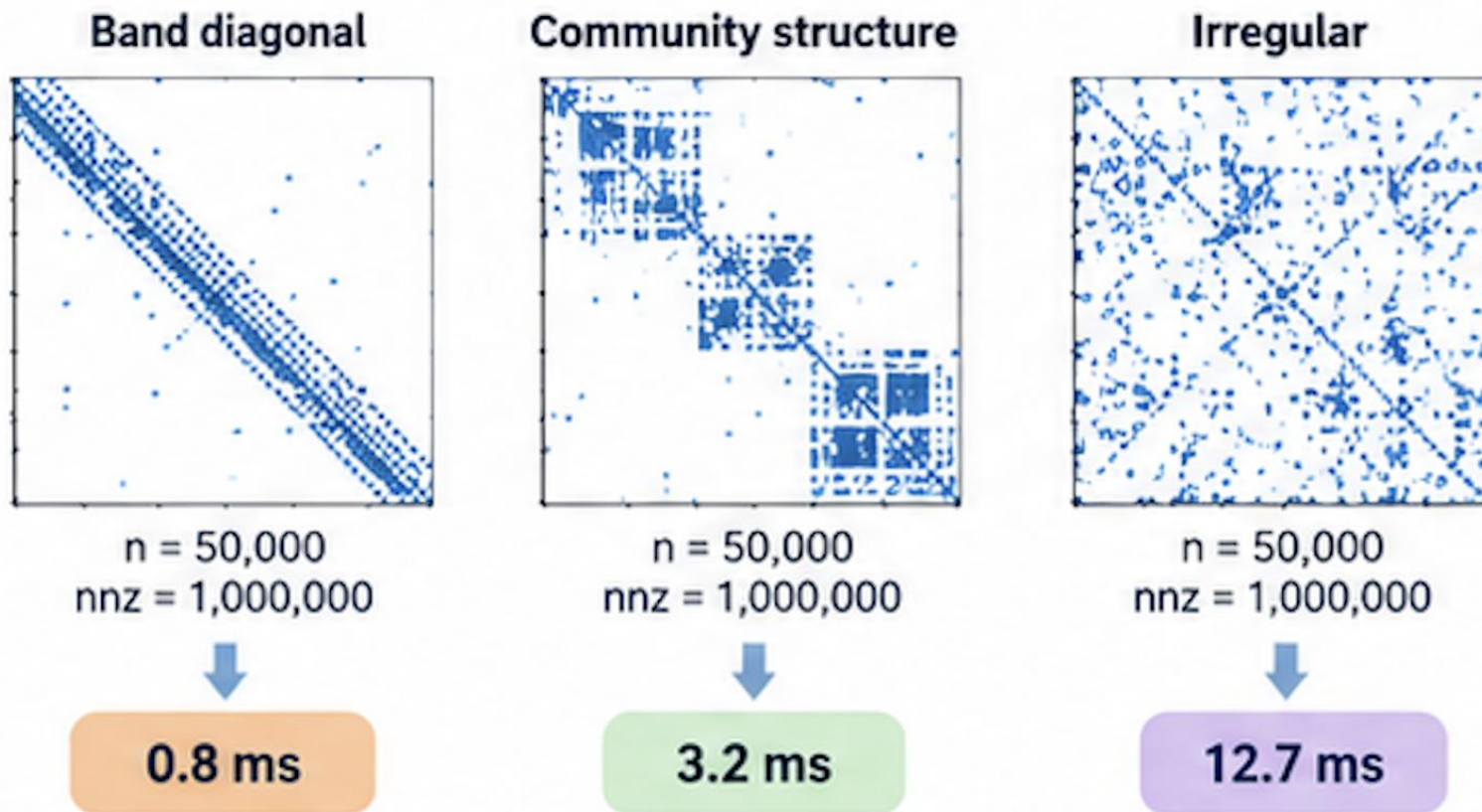


HPC Platforms

Learning Computational Signatures

Why can scientific computations of similar size behave very differently on the same hardware, and how can we make informed execution decisions without exhaustively testing every configuration?

- Representing the workload
- Predicting its behavior
- Selecting effective experiments and execution configurations.



SparseAtlas

SparseAtlas Corpus (Version 1)



1,945 matrices
(1,743 core + 202 Tier A)



203 groups
91 kinds



58.9 billion
nonzeros (NNZ)



Multiple data types
and symmetry classes

SparseAtlas is a characterized benchmark and research resource for understanding how sparse-matrix structure influences computational performance.

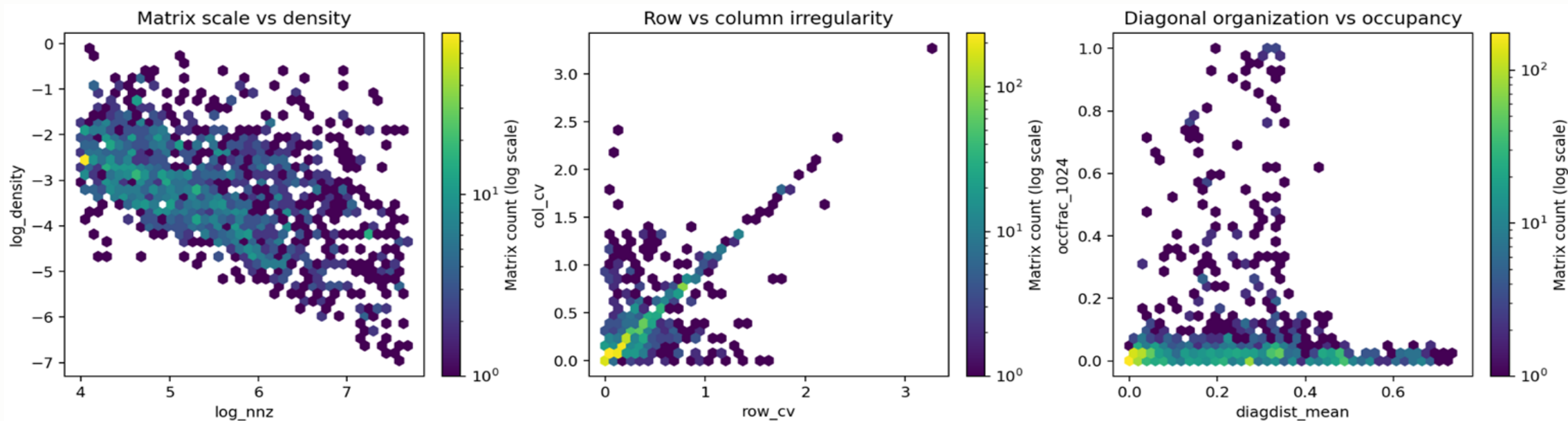
It combines structural matrix representations with measurements across GPU platforms: 7 independent runs, 4 GPUs, 3 architectures (Tesla P100 PCIe Pascal, Tesla V100 SXM2 Volta, GeForce RTX 5070 Ti Blackwell), resulting in 13615 measurements.

It is intended to be used to:

- Predict performance
- Select representative experiments
- Study portability across computing systems.

Structural Coverage

What computationally meaningful regions of sparse-matrix space does SparseAtlas represent, how densely are those regions sampled, and where are the limitations?



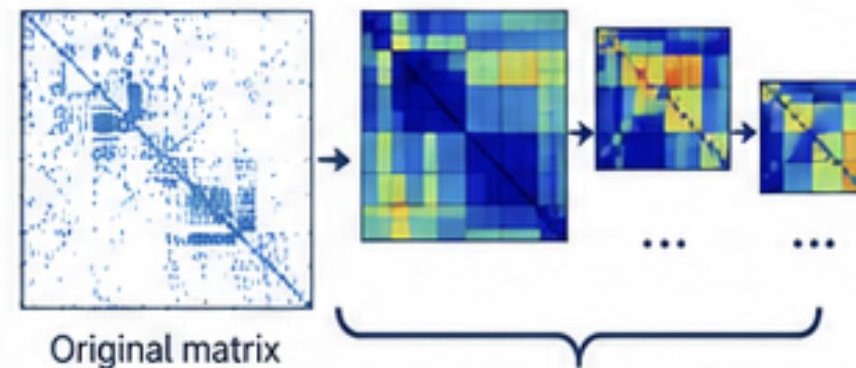
Pairwise relationships between structural characteristics. Joint distributions reveal which combinations are common, rare, or absent from this corpus.

Structural Descriptors vs Performance Prediction Across GPUs

Across seven GPU benchmark runs:

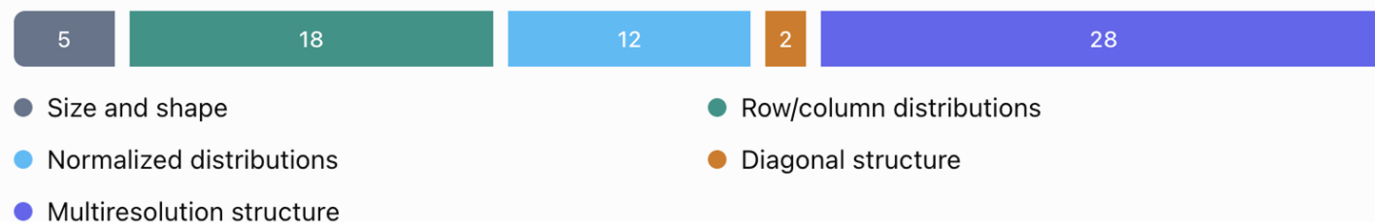
- Structural descriptors improve runtime prediction relative to conventional size/shape features.
- A small number of target-GPU measurements substantially improves cross-hardware predictions.
- Structural isolation helps identify matrices with greater prediction error, including when transferring across GPUs.

Multiresolution Structural Representation

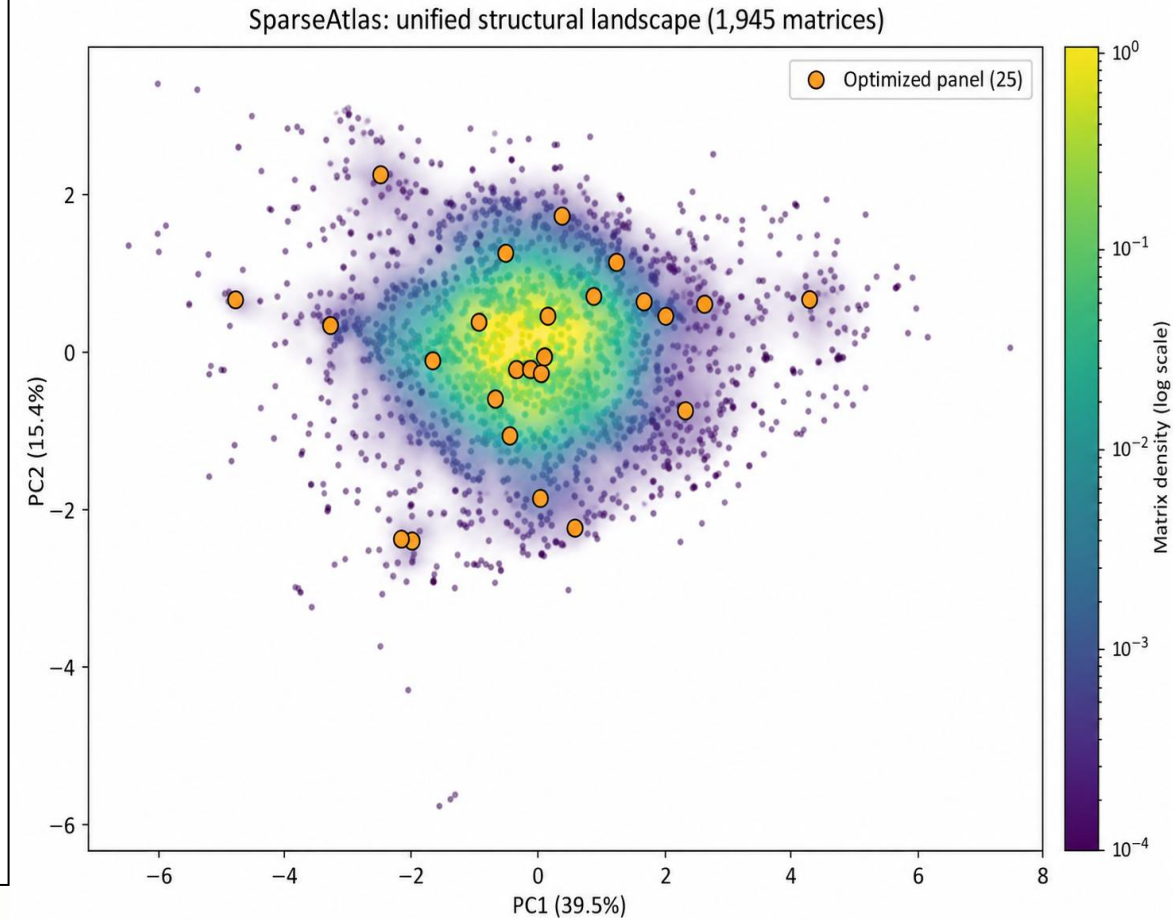
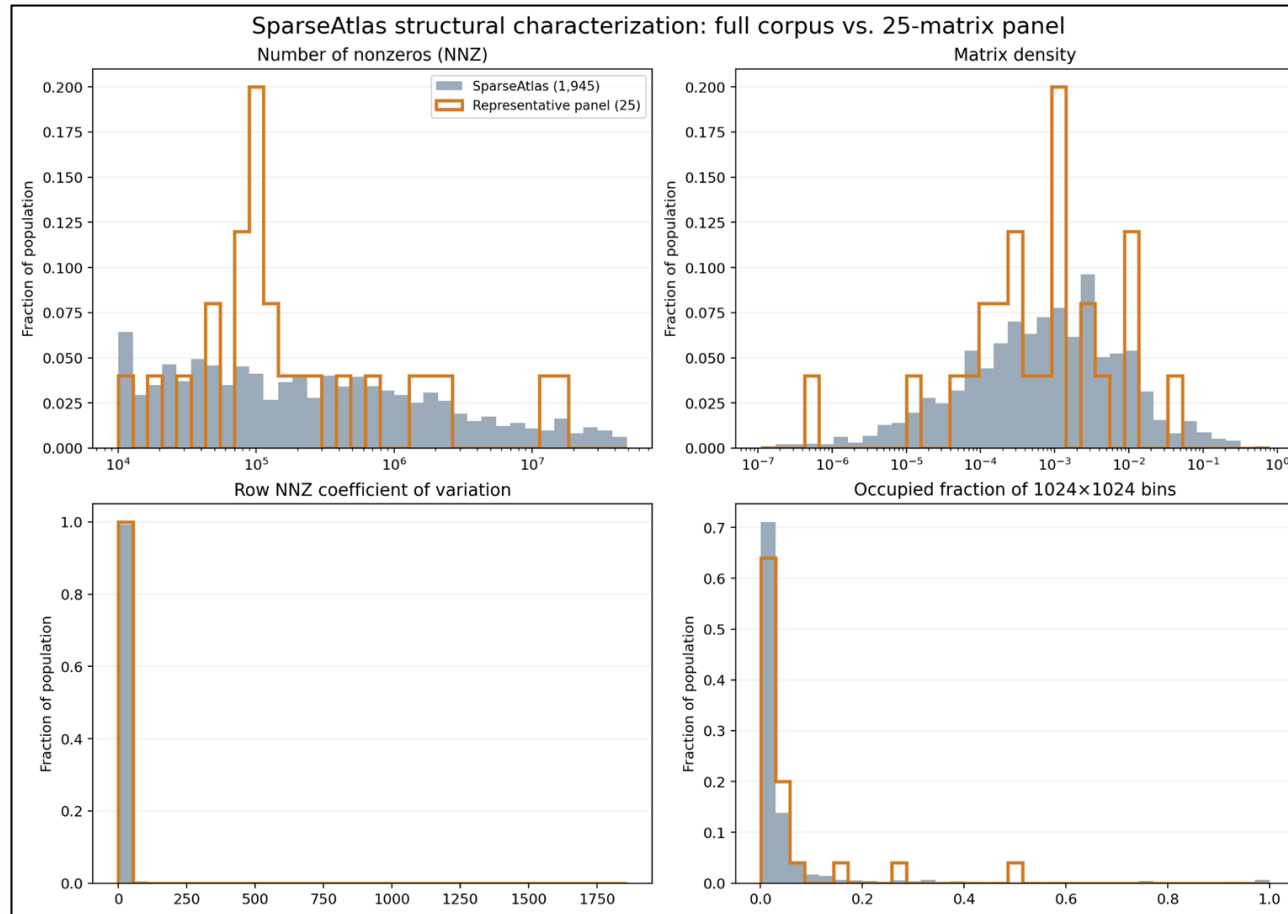


65 descriptors per sparse matrix

Feature composition from the SparseAtlas release



Optimized 25-matrix Panel for Selection of Experiments



Roadmap of Performance Expansion

Stage	Measurements	Scientific question
Single GPU	SpMV latency, throughput	How expensive is local sparse computation?
Partition analysis	Halo volume, remote references, load imbalance, communication partners	How does matrix structure translate into distributed work?
Multi-GPU	Compute/communication breakdown, synchronization, speedup	Which structures scale within a node?
Multi-node	Communication time, network volume, efficiency, rank placement	Which structures are sensitive to distributed hardware topology?
Solvers	Setup time, iterations, residual histories, convergence, total runtime	How does matrix structure affect complete numerical algorithms?

Few-Shot HPC Slowdown Prediction

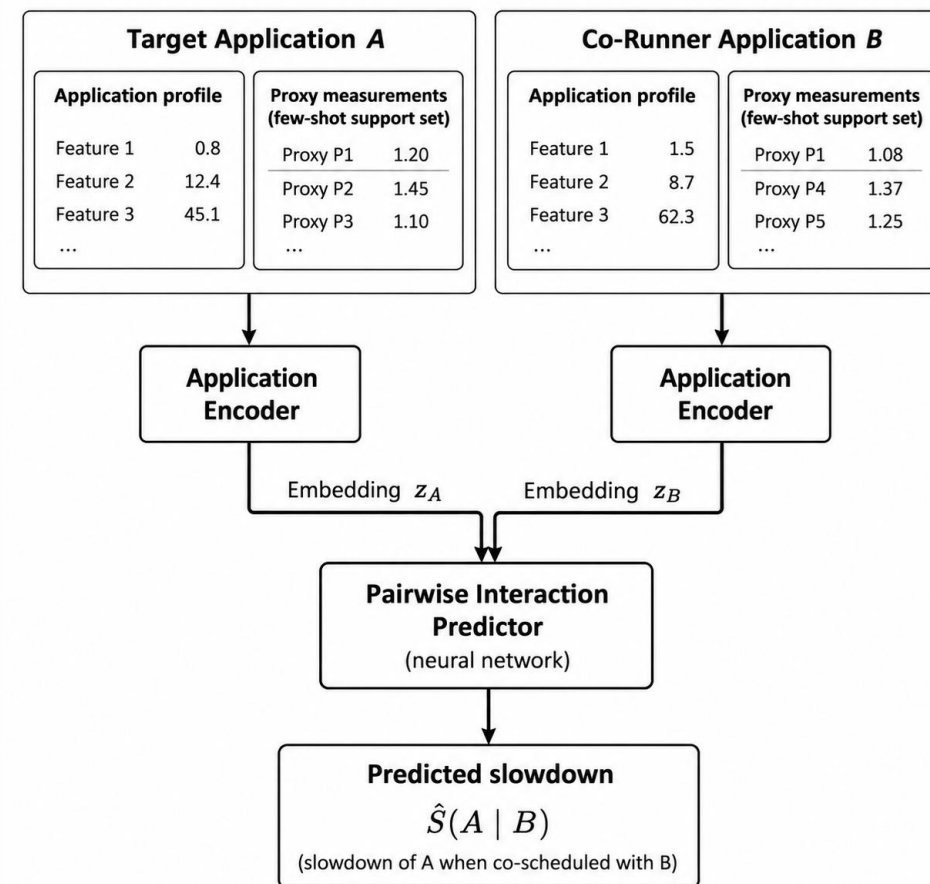
The goal is to predict how much a target application slows down when it runs at the same time as another application.

For example, given applications (A) and (B), we want to predict the slowdown of (A) when (A) is co-scheduled with (B).

We do not want to run every possible application pair. Instead, we characterize each application using two sources of information:

- Its normal execution profile, such as memory bandwidth, cache misses, IPC, FLOP rate, communication activity, or I/O activity.
- A small number of co-runs with carefully selected proxy applications.

For more information: UNM Student Akhil Alasandagutti



Agents Plane

Interpret specifications

Generate experiments

Operate within guardrails

Adapt based on results

Application specification

Experiment Planning and Orchestration
(existing tools - optional)

Analysis and Adaptation

Multi-modal
Embedding Space

Application/System

Example (TOML)

```
description = "AMG Solver"
docs = ["docs/amg_docs.pdf"]
[run]
template = "amg -P {Px} {Py}
          {Pz} -n {nx} {ny} {nz}
          -problem {problem}"

[run.params.problem]
type = "choice"
choices = [1, 2]
description = "Problem number"
```

- Application documentation
- Parameter space and allowed values
- Resource and execution constraints

Benchmark

benchmark

Experiment definition and reproducibility

- System specifications
- Application configurations
- Spack environments
- Parameter sweeps
- Reproducible experiments

Snakemake

snakemake

Workflow construction and data pipelines

- Input generation
- Preprocessing
- Simulation workflows
- Postprocessing
- Data management

Merlin

MERLIN

Experiment orchestration and campaign management

- Parameter sweeps
- Task dependencies
- Campaign monitoring
- Fault tolerance
- Iterative execution

Flux

Flux

Resource management and scheduling

- Resource allocation
- Job scheduling
- Task placement
- Multi-node execution
- Integration with schedulers

Performance Models

- Predict performance
- Quantify uncertainty
- Identify informative experiments

Experiment results

- Runtime and scalability
- System telemetry
- Metadata and logs
- Structured dataset

Data Collection



Applications



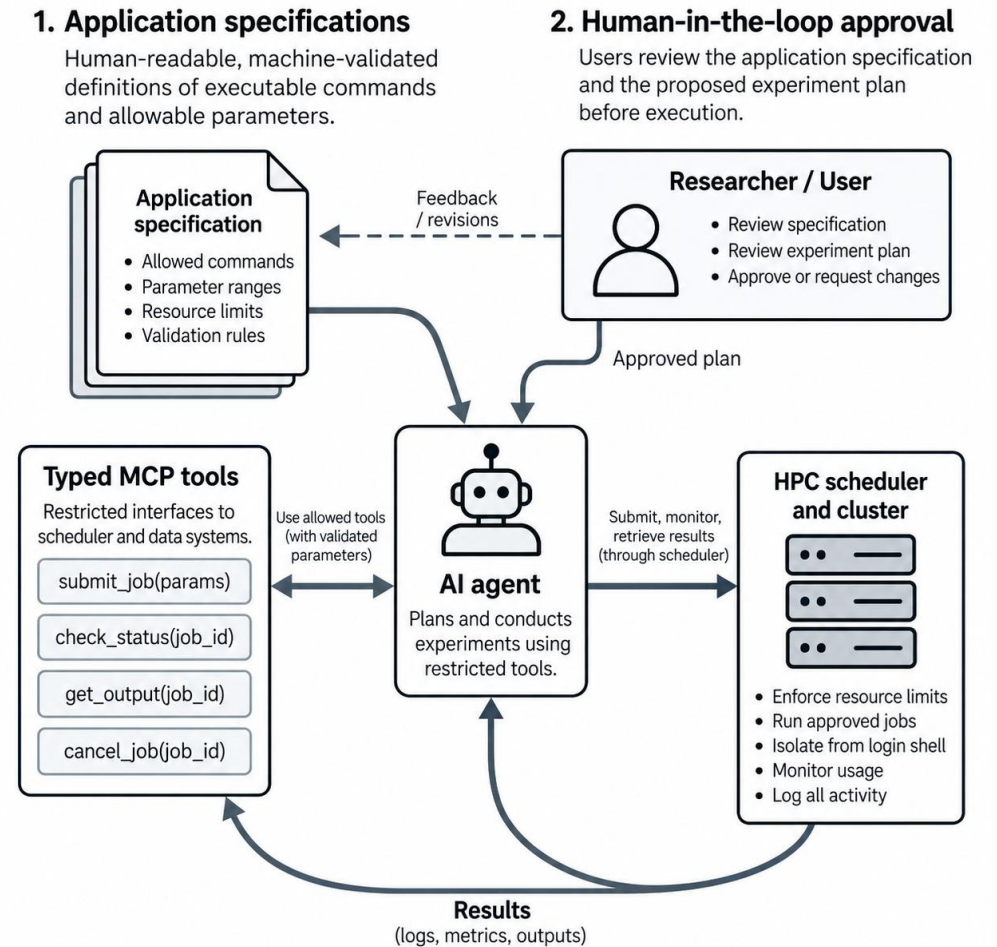
HPC Platforms

Controlled Autonomy for HPC Experimentation

How can we allow AI agents to conduct experiments on HPC systems without giving them unrestricted control over expensive, shared computing resources?

- **Application Specifications:** Human-readable, machine-validated definitions of executable commands and allowable parameters.
- **Typed MCP tools:** Agents interact with the scheduler through restricted interfaces rather than arbitrary shell commands.
- **Human-in-the-loop approval:** Users review application specifications and experiment plans before execution.

For more information: UNM Student Nico Livingstone



The Challenge Is Knowing Where to Look Next

What do we know?

Learned representations and performance predictions

What can we learn?

Uncertainty, distribution shifts, and missing experiments

How should we react?

Experiment selection, resource allocation, and execution decisions

Agents Plane

Interpret specifications

Generate experiments

Operate within guardrails

Adapt based on results

Application specification

Experiment Planning and Orchestration
(existing tools - optional)

Analysis and Adaptation

Multi-modal
Embedding Space

Application/System



Example (TOML)

```
description = "AMG Solver"
docs = ["docs/amg_docs.pdf"]
[run]
template = "amg -P {Px} {Py}
          {Pz} -n {nx} {ny} {nz}
          -problem {problem}"

[run.params.problem]
type = "choice"
choices = [1, 2]
description = "Problem number"
```



Application documentation



Parameter space and
allowed values



Resource and execution
constraints

Benchmark



Experiment definition
and reproducibility

- System specifications
- Application configurations
- Spack environments
- Parameter sweeps
- Reproducible experiments

Snakemake



Workflow construction
and data pipelines

- Input generation
- Preprocessing
- Simulation workflows
- Postprocessing
- Data management

Merlin



Experiment orchestration
and campaign management

- Parameter sweeps
- Task dependencies
- Campaign monitoring
- Fault tolerance
- Iterative execution

Flux



Resource management
and scheduling

- Resource allocation
- Job scheduling
- Task placement
- Multi-node execution
- Integration with schedulers



Performance Models

- Predict performance
- Quantify uncertainty
- Identify informative experiments



Experiment results

- Runtime and scalability
- System telemetry
- Metadata and logs
- Structured dataset

Data Collection



Applications



HPC Platforms

Conclusions and Acknowledgements

Understanding computation is the foundation for intelligent HPC.

- Learning meaningful representations of scientific workloads enables us to reason about computational behavior and performance across execution environments.

Autonomy requires guardrails, not unrestricted control.

- Agentic HPC can turn predictions and decisions into actions, but trustworthy execution requires explicit constraints, validated interfaces, and human oversight.

Next steps:

- Build SparseAtlas as a benchmark corpus that captures structural, computational, and communication behavior across platforms.
- Continue advancing AgHPCs autonomous capabilities while building rigorous safety guardrails and human accountability.

This work was performed with support from the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under the Predictive Science Academic Alliance Program Award DE-NA0004267

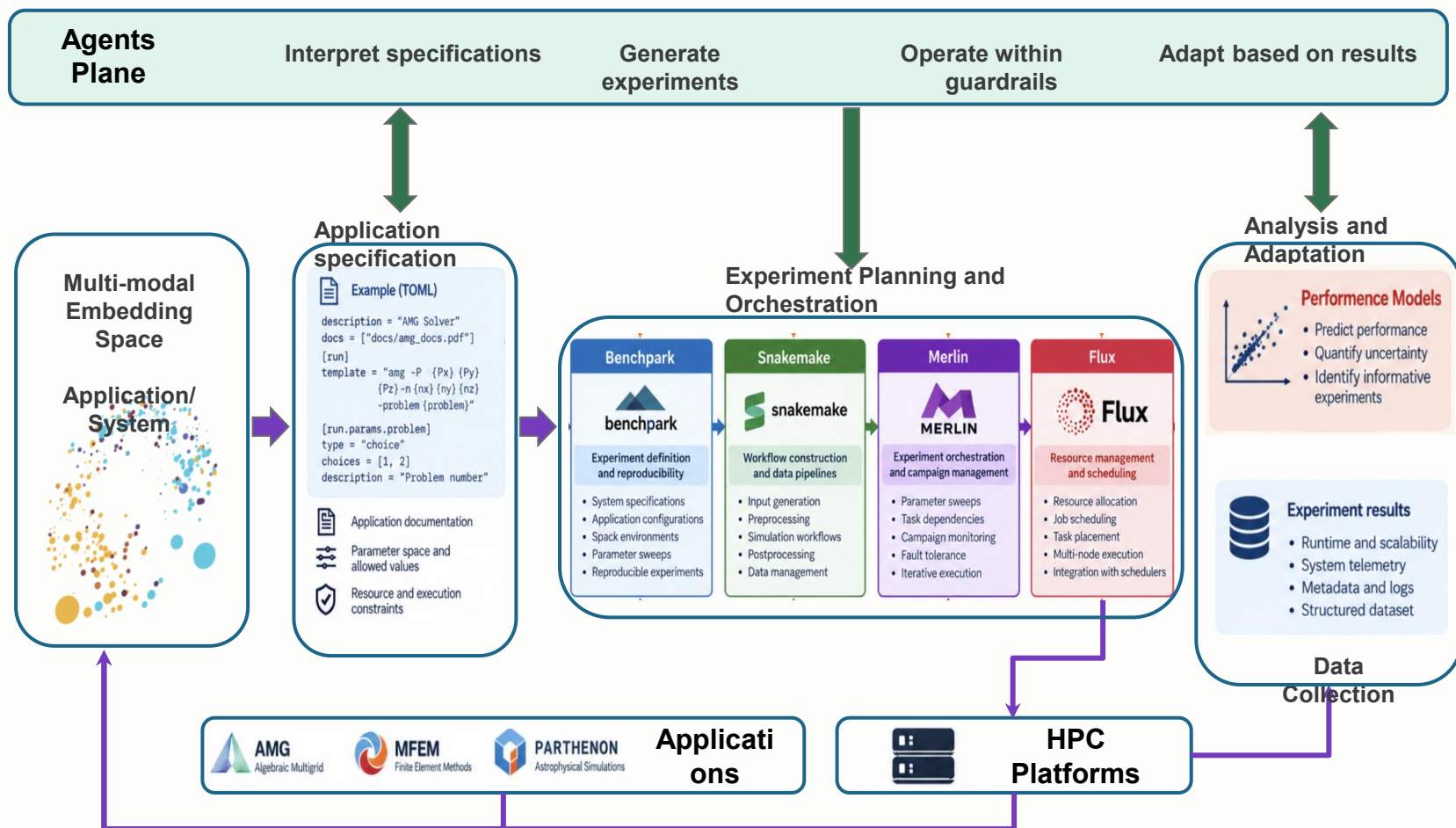


Toward Predictive and Adaptive Scientific Computing

Performance Tuning and Software Infrastructure Vision



One Software Infrastructure: Where This Work Fits



Analysis & Adaptation

Autotuning and decision support: predict, quantify uncertainty, pick the next experiment

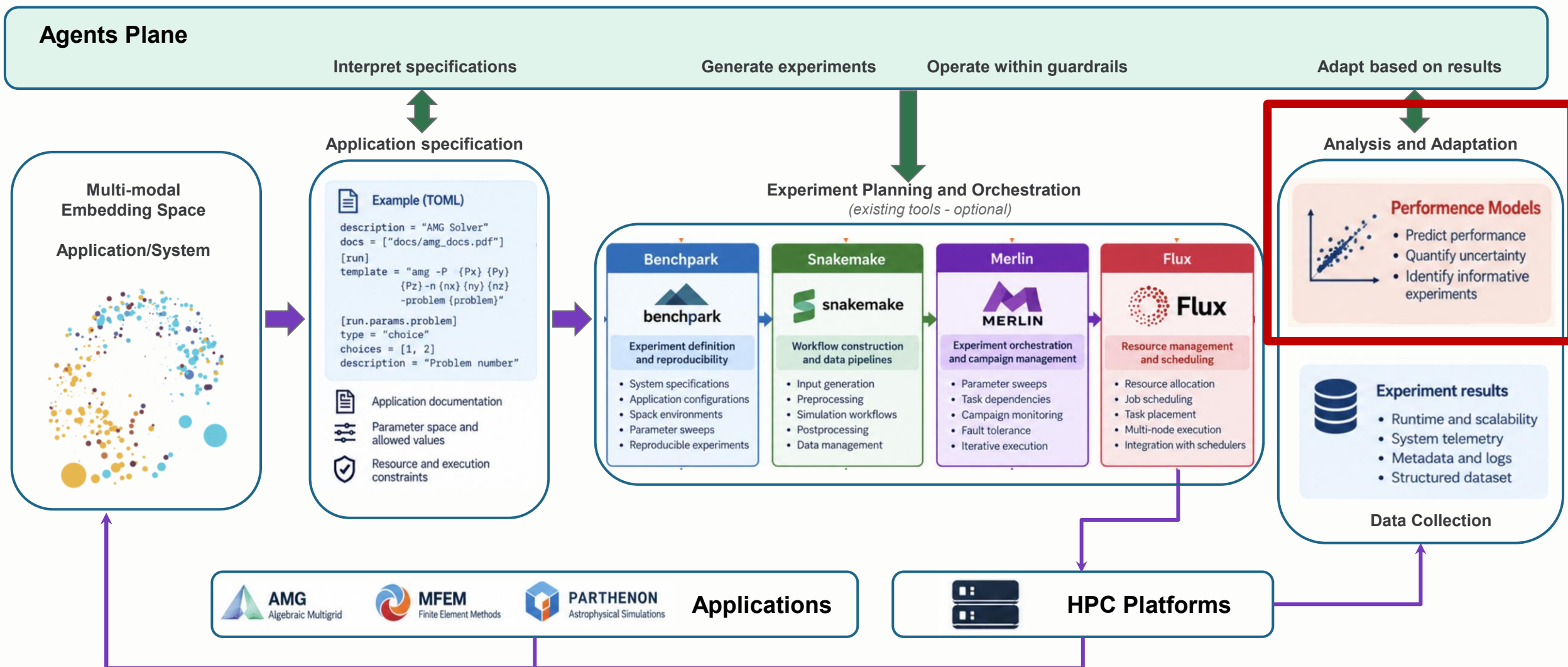
Data Collection

Tuning sweeps and cross-layer telemetry feed one structured dataset

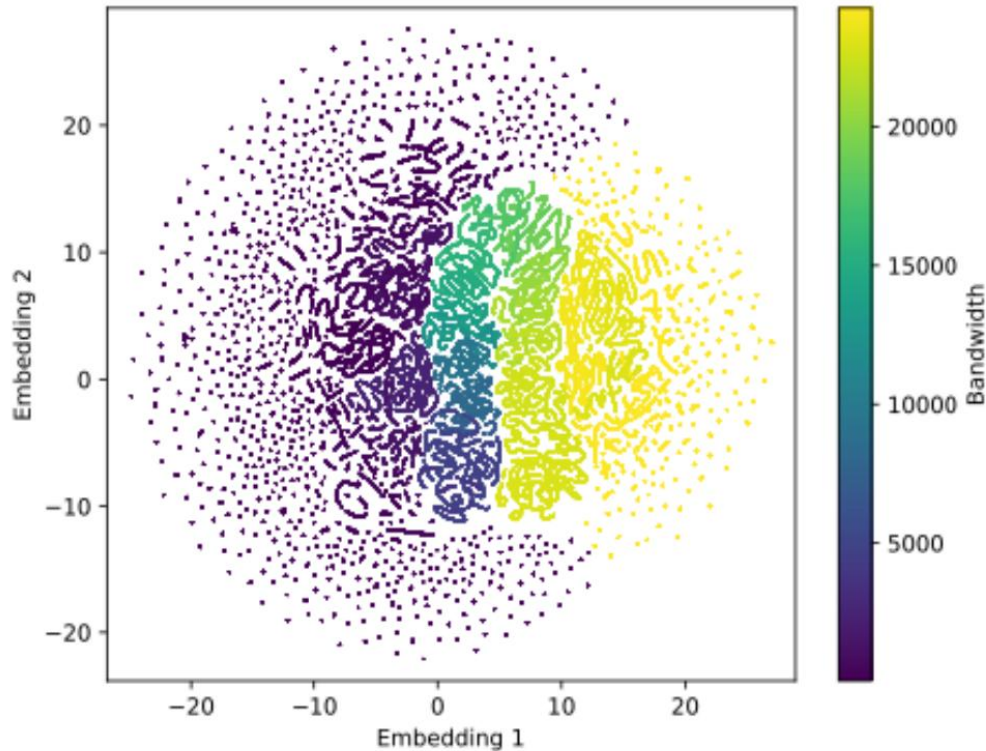
Agents Plane

AI practice: shared skills, typed MCP tools, guardrails

One Software Infrastructure: Where This Work Fits



Jointly Tuning Many MPI Parameters



UMAP embedding of MPI parameter settings, colored by bandwidth

Problem: System software includes dozens of tuning parameters which impact application performance

Question: Can we find good settings for many **interacting** MPI tuning parameters at once?

Why it is hard

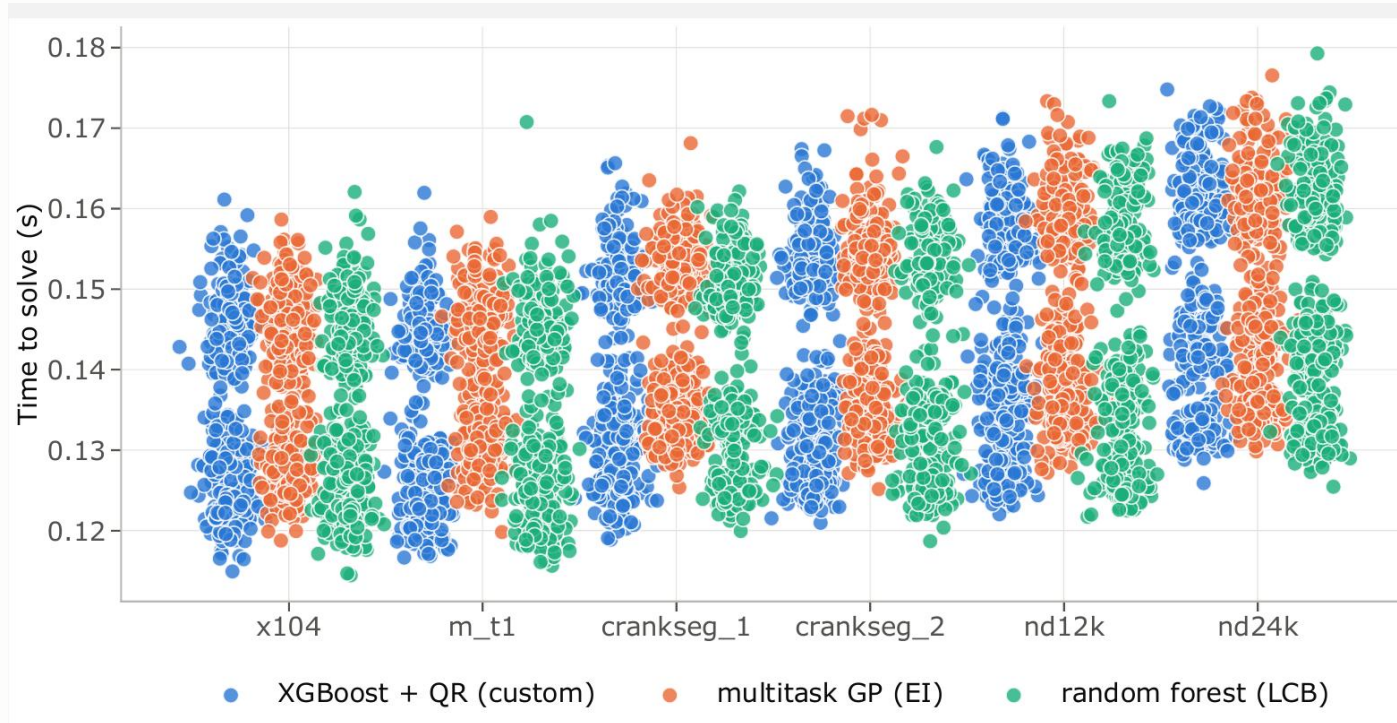
- Deep network stacks and several communication styles
- Heterogeneous CPU and GPU nodes
- Parameters interact, so one-at-a-time tuning misses optima

Approach One

- Manifold learning (UMAP) reduces the search space
- Derivative-free Bayesian optimization searches it, robust to noise

Shannon Kinhead: **details at the 3:30 poster session**

Autotuning Starts from Noisy Measurements



aCG on Tuolumne: time to solve across MPI configurations, three autotuning sessions

30%

run-to-run spread with the default configuration

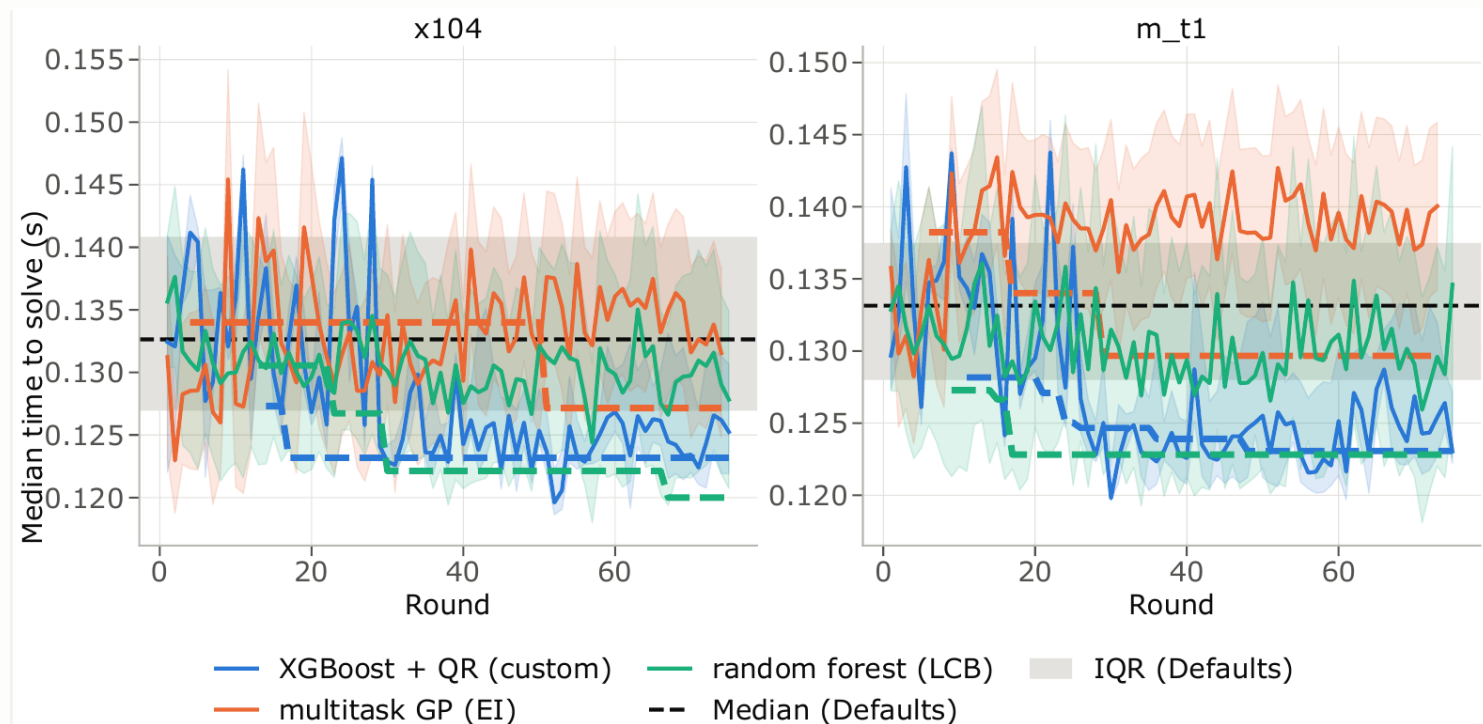
40%

spread when tuning intra-node GPU communication parameters

Approach Two: Integrate non-parametric uncertainty, not single measurements, into the tuning process

Xavier Barr: **details at the 3:30 poster session**

Bayesian Optimization of Cray MPICH Parameters



Solid: predicted median · dashed: best configuration found (≥ 5 runs) · 16 processes on 4 nodes

Approach

Quantile regression with XGBoost and a custom acquisition function, vs. a Gaussian process (EI) and a random forest (LCB)

Result

Finds better configurations than the GP, which assumes normally distributed solve times; matches the RF, with clearer explore and exploit phases

Xavier Barr: **details at the 3:30 poster session**

Decision Support: When Does an MPI Optimization Pay Off?

Question: When should an application adopt a communication optimization?

Why it is hard – this is a UQ problem

- Changing a production application is difficult
- Depends on details of communication pattern and system
- Lots of variance in runtime of real applications and benchmark

Approach

- PSAAP-III developed Vernier for capturing communication patterns
- Irregular communication benchmark as application analogue
- Adopt robust non-parametric statistics for decision support

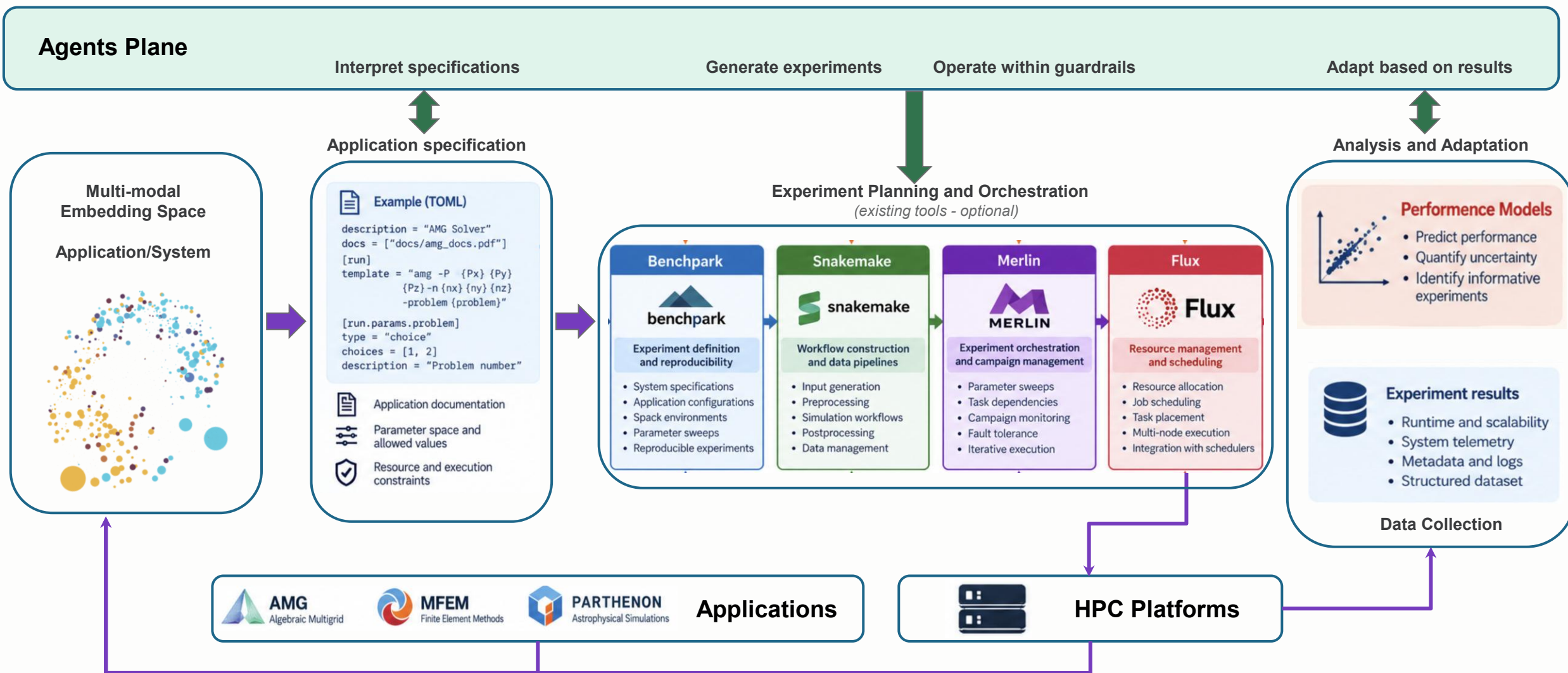
Nicholas Bacon: **details at the 3:30 poster session**

	LOCALITY wins by	one allocation	allocations mixed	mean of 40	worst of 40
crankseg_1, 2 nodes	2.01×	68%	51%	✓	✓
crankseg_2, 2 nodes	2.14×	68%	47%	✓	✓
m_t1, 16 nodes	1.07×	80%	51%	✓	✓
x104, 16 nodes	1.02×	32%	35%	✓	✓

Optimization	Wins when	Loses when
Stream-triggered halo exchange (Cabana)	4 GPUs per node, at every rank count	1–2 GPUs per node at the largest scales
Stream-triggered Kokkos Comm	Large halos or interiors: 0.92–0.97× MPI time	Small halo and interior: 1.7–1.8× ($\approx 13 \mu\text{s}$ per op)
Stream triggering in aCG	Ahead of RCCL	Behind host MPI overall (depends on matrix)
Earlybird collectives	Allgather below 1 KiB (1.2–2.6×); Alltoall from 1 KiB (1.7–4×)	Below system MPI at ≤ 64 B on Cray
Locality-Aware Neighbor Communication	Semi-sparse communication patterns (higher-level AMG matrices, selected CG matrices)	Fully sparse communication patterns (low-level AMG matrices, many CG matrices)

From the 1:00 communication talk and posters (Schafer, Suggs, Avans, Bacon, Wesley);

Vision: An Integrated Agentic System Software Adaptation Framework



Best Practices for AI Tools in Center Research

Everyone on the team uses AI assistants; we want that use to be **consistent, shared, and within Lab rules**. This also needs to be the groundwork for the infrastructure's Agents Plane

1 · DEFAULTS

Agent setup in template repositories

- Default agent instructions for each COMPASS template repo
- Shared skills, e.g. design-doc-writer, write-prompt
- MCP tools for center systems, e.g. a scheduler server

2 · EXPERIENCE

Collecting what works and what doesn't

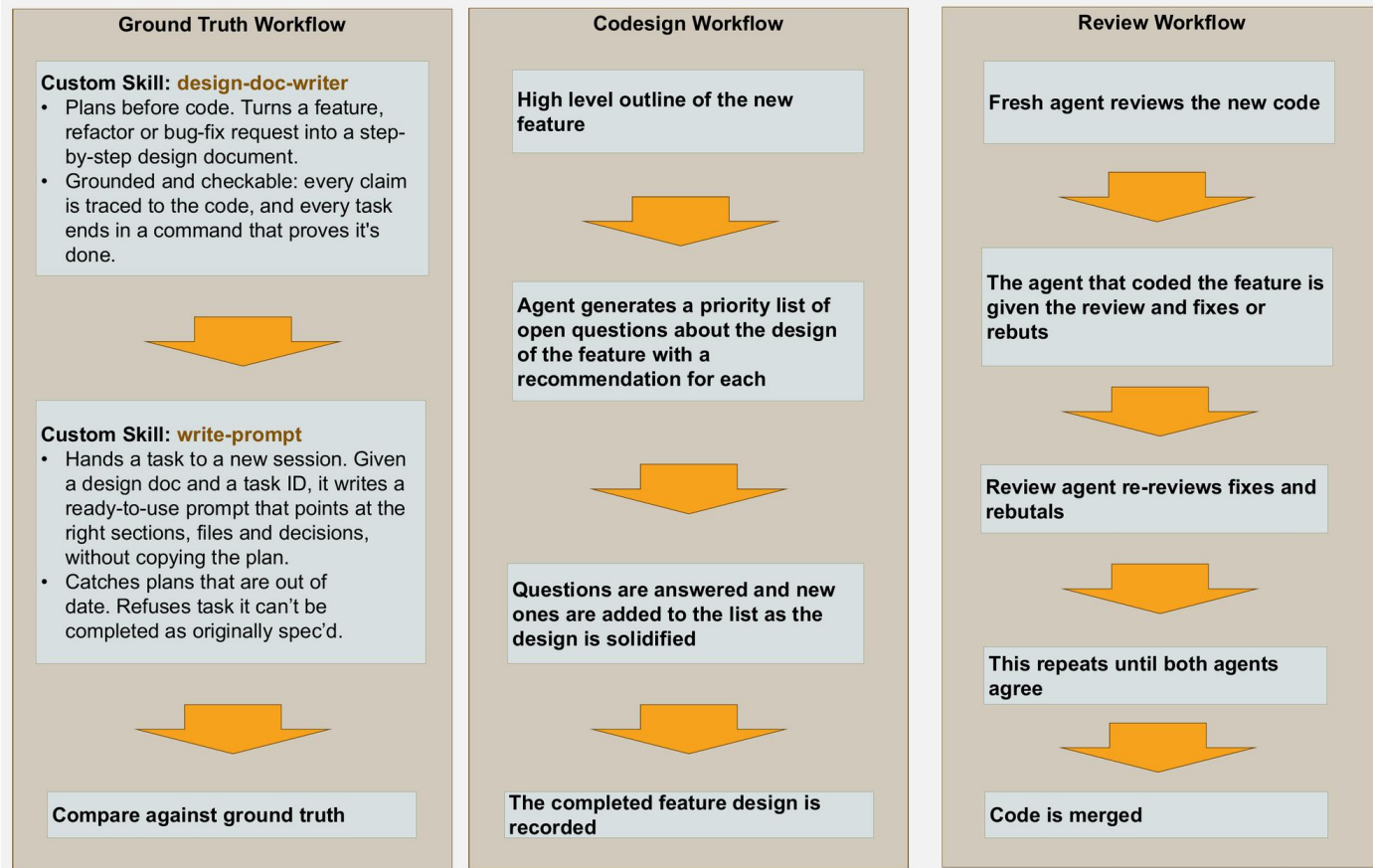
- Capture workflows and failures across students and sites
- Fold lessons into the defaults and the bootcamp

3 · RULES

Lab guidance for AI use on machines

- Review LANL, LLNL, and Sandia guidance on AI tools on Lab systems
- Map it to agent guardrails, system design, and repository defaults

Early Experience: Skills and Agent Workflows



What works

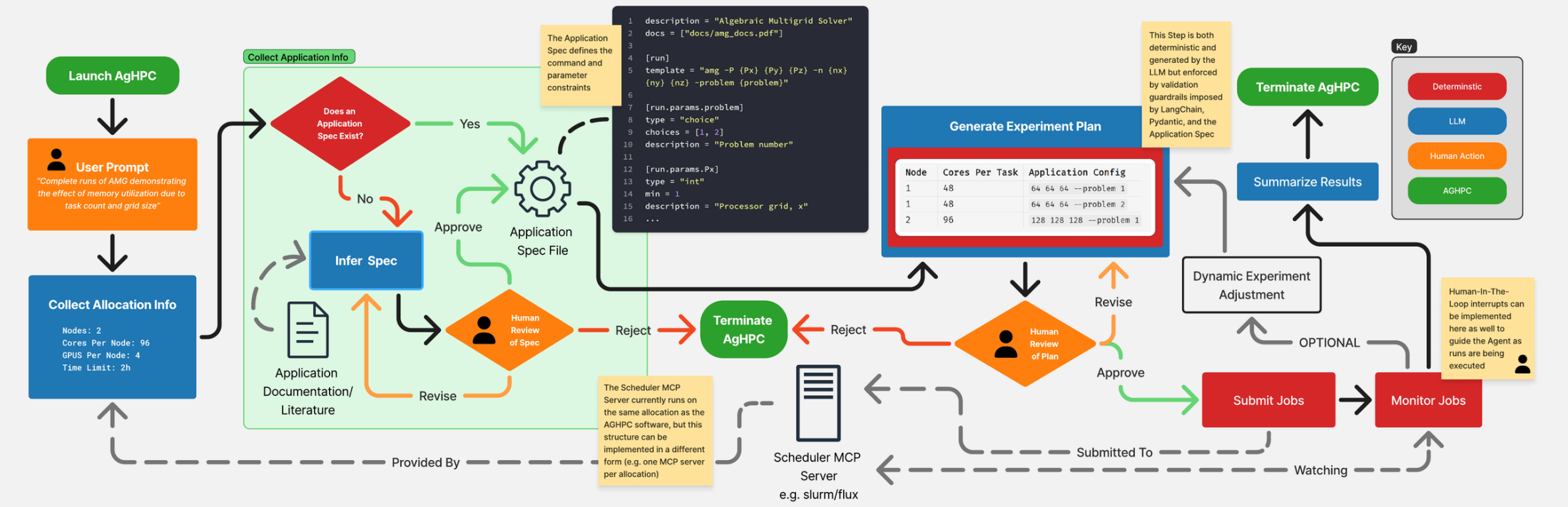
- Developing against a known ground truth
- Design docs before code; a fresh agent reviews
- More done in the same time: weak, not strong, scaling

Open questions

- How well must we understand the code, and the tests?
- Long comments, code bloat, simplest-route fixes
- Skills drift as models change every few months

Stewart and Hines, Workflows Using Artificial Intelligence (poster)

Agentic Workflows on HPC Systems Need Guardrails



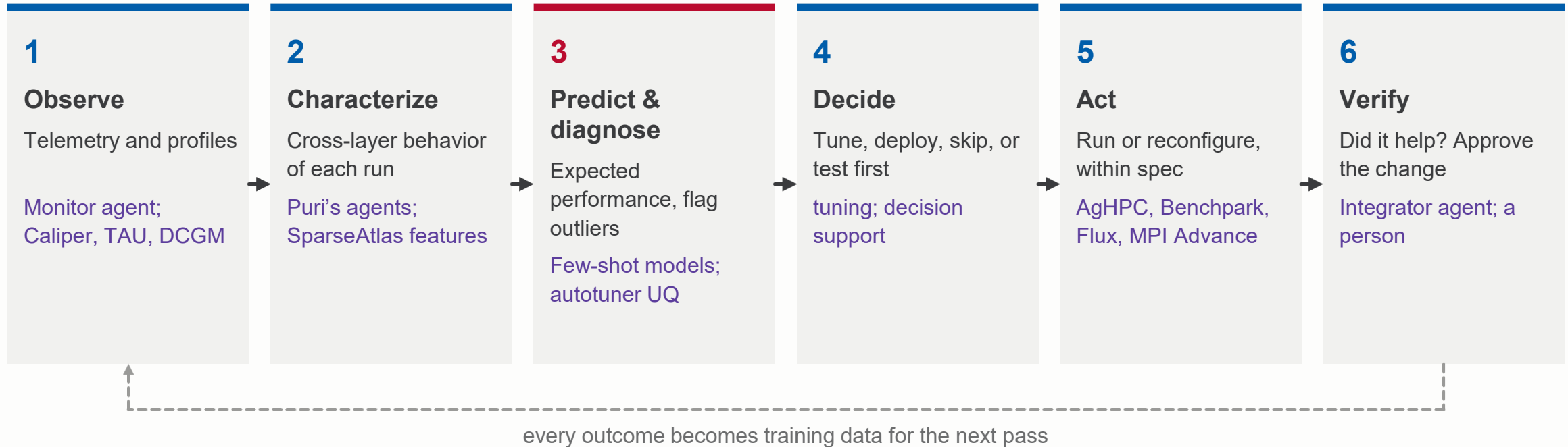
AgHPC's answer (Livingstone, Estrada)

Application specs bound every action; agents act only through typed MCP tools; a person approves the spec and plan before any allocation is used

To check against Lab guidance

Which assistants and models may run on or reach Lab systems; what code and data may be shared; approval and logging requirements

The Adaptation Loop, and Whose Tools Serve Each Step



Example: AI data analysis done as part of our decision support research **flagged an in-cast resource exhaustion problem with some communication patterns on Dane**. Diagnosing it, choosing a fix, and verifying it is the first end-to-end case.