RH RQ

Bringing great research ideas into open source communities

Jim Pfaendtner

Stop talking, start doing: how universities rise to the challenge of the AI revolution

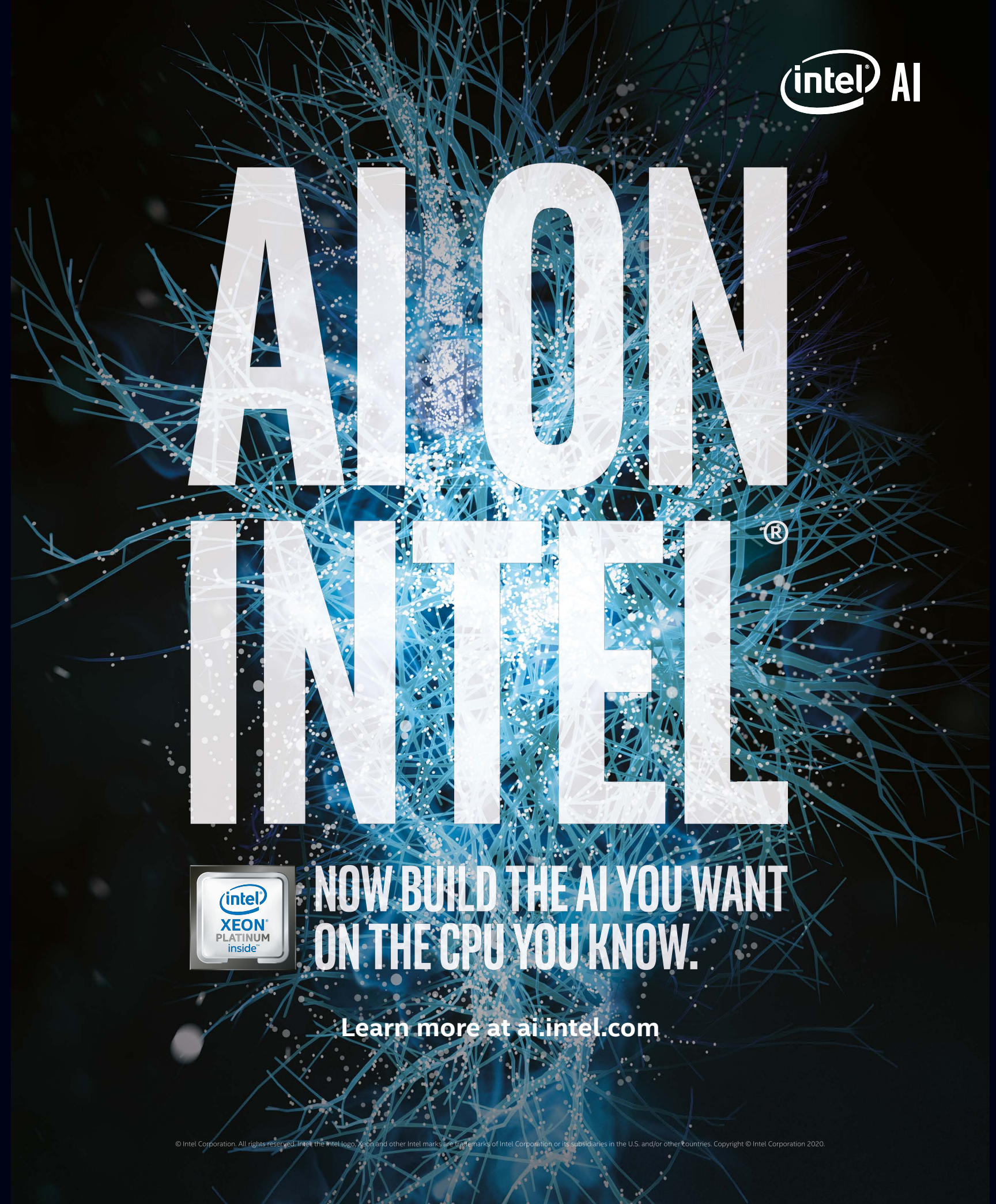
\+

Particle filtering for better LLM reasoning

Teaching code models to think without breaking their tools

Beyond the leaderboard: rethinking time-series foundation models

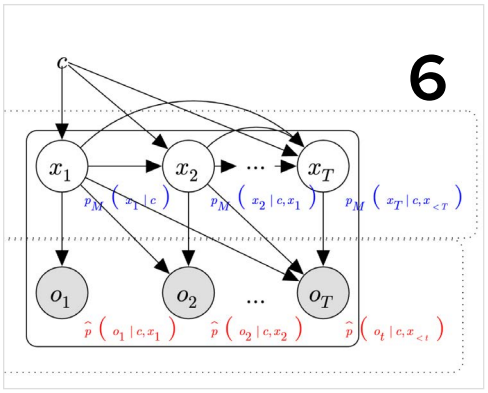
Red Hat Research Quarterly
Volume 8:1 | Summer 2026 | ISSN 2691-5278

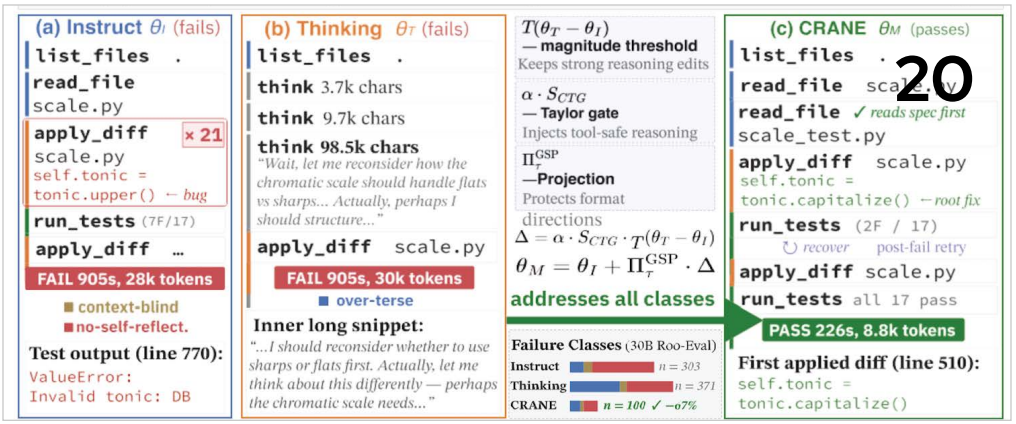


RESEARCH QUARTERLY

VOLUME 8:1

Table of Contents





Departments

Features

Red Hat

RESEARCH QUARTERLY

VOLUME 8:1

From the editor

The new research pipeline: open ecosystems are powering real AI progress

About the Author
Shaun Strohmer is the editor of the *Red Hat Research Quarterly*. She has worked as a writer and editor in academic publishing for nearly two decades, and since 2014 she has focused on software development, cybersecurity, and computer science.

by Shaun Strohmer

Not so long ago, industry research meant exploring ideas that were typically anywhere from 3 to 10 years out from actual product implementation (even faster in open source). If you've forgotten that definition, you're excused. The AI era has found researchers building express lanes and wormholes throughout the research-to-production pipeline.

Going faster doesn't have to mean cutting corners, however. On the contrary, I bring up this disruption because while putting this issue together I was struck by the diversity of research taking an open source approach precisely because we're all invested in making AI legitimately usable by doing things right. The Summer 2026 edition of RHRQ highlights many ways industry engages with research: Red Hat partnerships with individual universities, advanced PhD research guided in part by Red Hat engineers, cross-industry cooperation within an international benchmarking consortium, and open ecosystems like the AI Alliance. The key is open collaboration, breaking down walls between university researchers, industry engineers, and users so that everyone can move faster.

North Carolina State University Provost Jim Pfaendtner described the academic version of open source as maximizing "the velocity of dissemination." In his interview with Senior Director of Engineering for Red Hat Open Shift AI Sherard Griffin, Pfaendtner talks about how university-industry partnerships are a critical factor in keeping a collective foot on the accelerator. From preparing the next generation of engineers to transferring technology via university-founded startups or leveraging AI to transform engineering design, Pfaendtner describes a compelling vision for making a large land-grant university agile enough to work at the speed of AI. Read "Stop talking, start doing: how universities rise to the challenge of the AI revolution" to learn more.

In September 2025, we announced a collaboration among Red Hat, IBM Research, the AI Alliance, and the Mass Open Cloud to support several short-term (less than one year) research projects as part of the NAIRR pilot deep partnerships, and now they are bearing fruit. I'm excited to publish our first two articles about these ambitious projects in this issue, both led by PIs from Rensselaer Polytechnic Institute. Read "CRANE: teaching code models to think without breaking their tools" to discover a model-merging approach that transfers reasoning improvements from thinking models while preserving the reliable tool-use patterns of instruct models. If you're using an agent for software-engineering tasks that depend on reliable tool use, you understand why this matters. Or, consider the development of a Time-Series Data Agent: an agentic system that combines a numerically

focused time-series foundation model with a semantically focused large language model. This next frontier in forecasting models is the topic of "Beyond the leaderboard: rethinking time-series foundation models."

Also in this issue, we're spotlighting one of five papers from members of the Red Hat AI Innovation Team presented at NeurIPS 2025. In the RHRQ article "Particle filtering for better LLM reasoning," co-author Guangxuan Xu explains the method introduced in "Rollout roulette: a probabilistic inference approach to inference-time scaling of LLMs using particle-based Monte Carlo methods." In May 2026, a team of Red Hat performance engineers and a researcher at the Illinois Institute of Technology won the Best Industry Paper award for "Evaluating Kubernetes performance for GenAI inference: from automatic speech recognition to LLM summarization" at the SPEC International Conference on Performance Engineering; they share that work in the article "Kubernetes meets GenAI: evaluating performance for AI inference workloads." We've long followed work at Karlstad University focused on building the next generation of programmable networking—powered by Linux. Check out "Monitoring latency within the end host network stack" to get an early look at their latest development, **`netstacklat`**, an advanced latency monitoring tool the team will present at the ACM Internet Measurement Conference in October 2026.

In his column, "Building shoulders for giants to stand on," research engineering manager Robby Stahl observes that research in AI and AI-based research—in technical or systems problems, healthcare, or other domains—"is so much more than any person could accomplish in a lifetime." The infinite potential he describes is one of the things that makes research such an exciting place to be. At a time when the pace of change can be dizzying, research grounded in strong partnerships and communities provides wayfinding and guardrails to ensure that even as we're going faster and faster, we're still going moving in the right direction.

Red Hat

RESEARCH QUARTERLY

VOLUME 8:1

Feature

About the Author

Guangxuan (GX) Xu is a Principal Research Scientist and Agent Architect at Red Hat, previously a founding member of InstructLab. He holds an MS in Computer Science from UCLA. His research focuses on LLM alignment, inference-time scaling, and enterprise AI agent systems.

Particle filtering for better LLM reasoning: a different approach to inference-time scaling

Researchers are proving that small open source models, using a well-established algorithm, can compete successfully with proprietary frontier models.

by Guangxuan (GX) Xu

Large language models continue to improve through larger architectures and more training data, but recent evidence suggests diminishing returns from scaling model size alone. A complementary approach—inference-time scaling (ITS)—allocates more computation during inference to improve performance without increasing model parameters. Existing ITS methods such as beam search and diverse verifier tree search (DVTS) treat inference as a search problem guided by a process reward model (PRM), a separate neural network that scores each intermediate reasoning step. But these methods trust the PRM deterministically, pruning any path that scores poorly—even if the score is wrong. In our NeurIPS 2025 paper "Rollout roulette: a probabilistic inference approach to inference-time scaling of LLMs using particle-based Monte Carlo methods," we show that a three-decade-old algorithm from signal processing—particle filtering (one of the Sequential Monte Carlo methods)—offers a principled alternative that treats reward scores as uncertain evidence rather than ground truth.

THE FILTERING PROBLEM

Imagine a cruise ship navigating through dense fog. The ship has a GPS sensor, but it is noisy—each reading scatters around the true position by some unpredictable amount. If you plot the raw GPS readings, you get a jagged, wandering path that only loosely resembles the ship's actual trajectory.

Here is the surprising part: particle filtering can track the ship more accurately than the GPS sensor itself, by learning from the same sensor data. In an example simulation we ran—a target moving along a 2D spiral over 200 time steps with Gaussian sensor noise (standard deviation 0.6)—a particle filter with 500 particles and a velocity model achieved a mean tracking error of 0.46, compared to 0.76

for the raw sensor (see **Figure 1**). That is a 39% improvement, using only the same noisy readings as input. This ability—producing better estimates than the sensor it relies on—is why particle filtering appears in robot localization, submarine navigation, and aircraft tracking.

HOW PARTICLE FILTERING WORKS

You want the true position of the ship, but you cannot observe it directly. You have two imperfect tools: a motion model that generates candidate positions for where the ship might be next, and a GPS sensor that scores each candidate. The algorithm repeats three steps at each time step:

1. **Propagate:** push each particle (candidate position) forward using the motion model.

2. **Score:** weight each particle based on how well it matches the latest sensor reading.

3. **Resample:** stochastically select particles for the next round. High-scoring particles are more likely to survive, but low-scoring ones are not guaranteed to die.

Why does this beat the sensor? Because the sensor is noisy, and particle filtering treats its readings as uncertain evidence rather than ground truth. If you deterministically kept only the top-K particles by sensor score at each step, the population would collapse into near-identical clones within a few steps—locked onto wherever the GPS noise happened to point. One bad reading, and the entire population drifts off course with no way to recover.

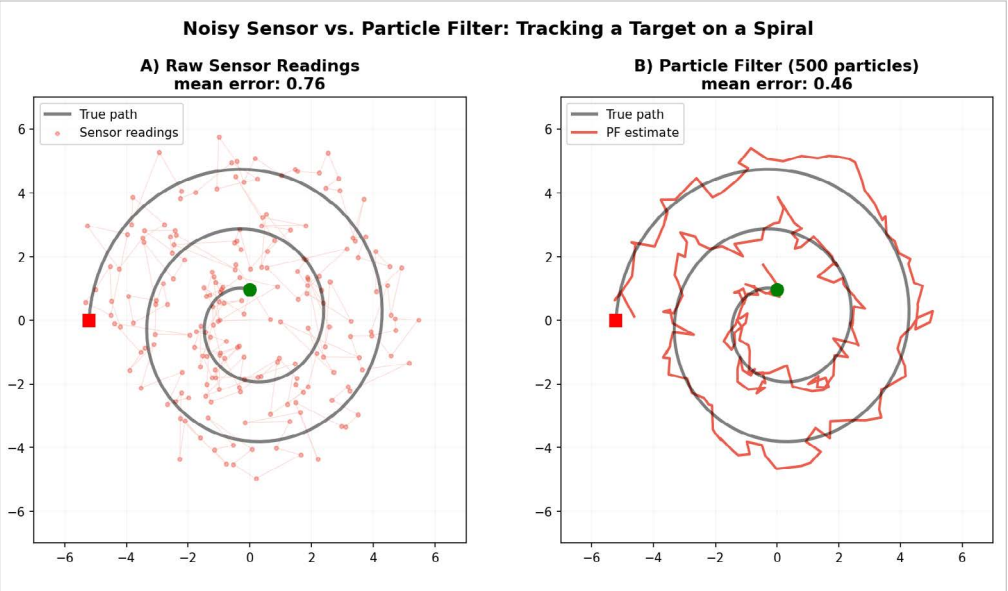


Figure 1. *Left: raw sensor readings scatter around the true path. Right: particle filter (500 particles) tracks the true path with 39% lower error, using only those same noisy readings as input.*

Stochastic resampling prevents this. High-scored particles are favored but not guaranteed to survive; low-scored particles can persist. A particle that looks mediocre now might be the only one near the true position three steps later. The population stays diverse enough to recover from scoring errors, rather than committing prematurely to whatever the noisy sensor happens to favor at any given step.

PARTICLE FILTERING FOR LLM REASONING

Now replace the cruise ship with a hard math problem. You want the correct solution, but you cannot generate it directly—if you could, the problem would already be solved. You have two imperfect tools, structurally identical to the filtering setup above.

The LLM serves as the transition model: given a partial solution, it generates the next candidate reasoning step. A process reward model (PRM) serves as the emission model: it scores each intermediate step for correctness, like a grader reading the LLM's work line by line. But the PRM is noisy. In **Figure 2** (overleaf), we show a real example where the PRM assigns a score of 0.9453 to the first step of a wrong answer and only 0.5156 to the first step of the correct answer. Any method that trusts these scores deterministically would immediately prune the correct solution.

This is exactly what beam search does. It keeps the top-K highest-scored partial solutions at each step and discards the rest. One bad score early on, and the correct path is gone permanently. Best-of-N takes a different but equally brittle approach: generate N complete solutions independently, then pick whichever the PRM scores

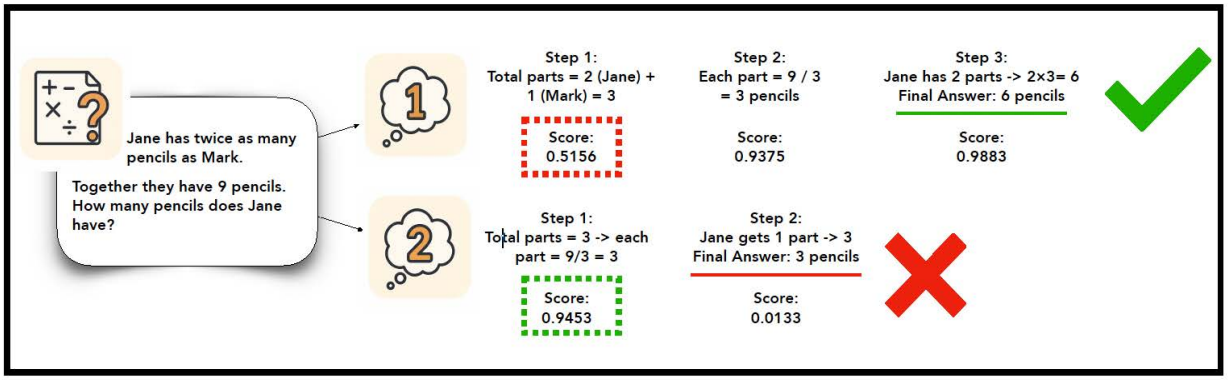


Figure 2. *The scores/evaluators for each step are approximate values that could be wrong. In this mathematics example, the first step that scored high ends up giving a wrong solution. And the first step that only scored 0.51 ends up producing the right answer. Particle filtering works because it prevents early pruning that overfits the biases of imperfect evaluators.*

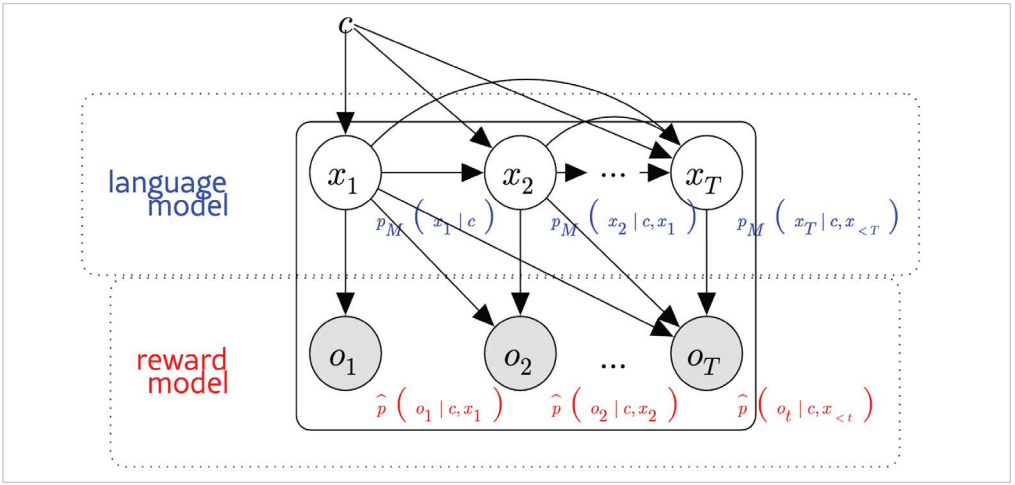


Figure 3. *State-space model for inference-time scaling.* c *is a prompt,* x1, . . . , xT *are LLM outputs, and* o1, . . . , oT *are "observed" acceptances from a reward model. We estimate the latent states conditioned on* ot = 1 *for all* t.

highest overall. It treats the reward model as ground truth and has no memory across reasoning steps—each solution is produced in isolation.

Our method formulates ITS as approximate posterior inference over a state-space model (**Figure 3**). Each particle is one partial reasoning path. At each reasoning step, we propagate (extend each path with the LLM), score (evaluate with the PRM), and resample (select paths via softmax-weighted stochastic sampling). The softmax resampling is the critical difference from beam search: low-scored paths can survive, and high-scored paths are favored but not guaranteed. This preserves population diversity and allows recovery from noisy early scores.

The parallel to classical filtering holds precisely. Just as the particle filter tracks a ship's position more accurately than the GPS sensor it relies on, our method finds correct solutions more reliably than the noisy PRM it uses for scoring. With prefix caching, the runtime is comparable to generating N full completions independently.

RESULTS

We performed our experiment on the MATH500 dataset, a set of 500 complex math problems used in the AI community to evaluate the mathematical reasoning capabilities of language models. Small open source models combined with particle filtering compete directly with frontier proprietary models. **Figure 4** shows the outcomes. Qwen2.5-Math-1.5B—a model small enough to run on modest hardware—paired with particle filtering achieves 84.6% on MATH500 with a budget of 32 generations, surpassing GPT-4o (76.2%) with a budget of only 4. At the 7B scale, Qwen2.5-Math-7B with particle filtering reaches 87.7% on MATH500, matching o1-preview (87.0%) at budget 32. On AIME 2024,

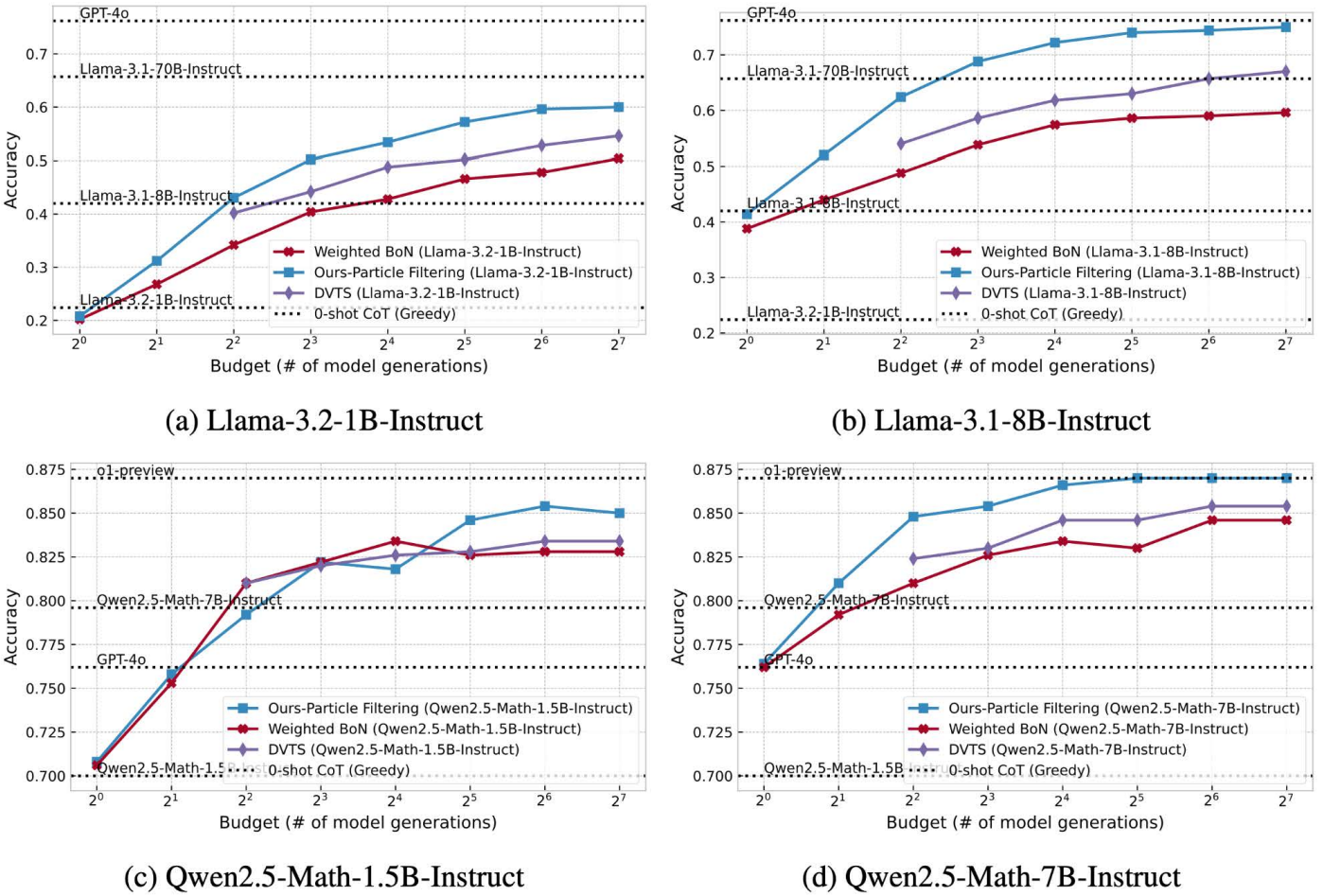


Figure 4. *Accuracy vs. compute budget across four models. Particle filtering (blue) consistently outperforms Weighted BoN (red) and DVTS (purple), scaling 4-16x faster.*

a notoriously difficult competition benchmark, it achieves 10/30 compared to 7/30 for beam search.

Particle filtering also scales more efficiently than alternatives. DVTS, the next best competitor, requires a budget of 32 to match what particle filtering achieves at budget 8—a 4-16x efficiency advantage. On non-math-specialized models, the gains are equally clear: Qwen2.5-1.5B (a general-purpose model) reaches 79.3% on MATH500 with particle filtering versus 76.2% for beam search and 76.6% for DVTS. The method also generalizes beyond mathematics: on FinanceBench, particle filtering achieves 70.33% versus 67.33% for beam search, and on NumGLUE Chemistry, 84.22% versus 80.47%.

WHAT THIS MEANS

For engineers running models in production, the takeaway is practical: a smaller model with principled inference-time scaling can match or exceed the performance of a much larger one at a fraction of the serving cost. And the algorithm enabling it is not new, but a well-understood technique from signal processing, applied to a problem where it turns out to fit remarkably well. Sometimes the best ideas in AI are the ones we already had.

Says who?

/Keep your options open
redhat.com/options

Red Hat

A AWS

B Azure

C Google Cloud

D All of the above

/Keep your options open
redhat.com/options

Red Hat

Stop **TALKING,** start **DOING**

How universities rise to the challenge of the AI revolution

*An interview with **Jim Pfaendtner***

*conducted by **Sherard Griffin***

Red Hat

Interview

The conversation around AI in higher education sometimes feels stuck in abstraction. When industry and academia work together, however, that conversation is immediately grounded in things like the transition to data-driven engineering, the necessity of academic agility, and the hard work of building infrastructure that is reliable, auditable, and open.

North Carolina State University Provost Jim Pfaendtner provides a front-row seat to how a major research institution navigates this transition. Jim brings a rare dual perspective to this challenge—a career rooted in the computational chemistry lab at the University of Washington and an administrative track that prioritizes the rapid adaptation of engineering curricula. (Note: This interview was conducted prior to Jim's appointment as Provost; he served as Dean of NC State's College of Engineering from 2023-2026.) The conversation is led by Sherard Griffin, Senior Director of Engineering for OpenShift AI at Red Hat. An NC State alum, Sherard brings a personal connection; having walked the Centennial Campus himself, he understands that the best partnerships aren't transactional—they are deeply integrated. This is the cornerstone of Red Hat's AI University Partnerships: creating a collaborative environment where industry and academia can turn experimentation into operational reality, to make AI scalable, secure, and accessible to everyone. —Shaun Strohmer, Ed.

About the Interviewer

Sherard Griffin has over 20 years of experience architecting and developing large-scale enterprise data and AI solutions. He has been at Red Hat since 2017 and is currently the Senior Director of Engineering for Red Hat OpenShift AI. He is also responsible for Open Data Hub, a community-driven open source project for building an AI-as-a-service platform on OpenShift. He works with hardware and software partners to build out an ecosystem of AI technologies optimized for Kubernetes, Open Data Hub, and OpenShift AI.

Sherard Griffin: You come from a background in both chemical engineering and computer science, which means you stepped into the computational side of science and the world of data and AI before it was cool. How did those worlds meet for you?

Jim Pfaendtner: I am in some ways a poster child for what we now call interdisciplinarity. I got my undergrad in chemical engineering, and I worked at a hands-on job at 3M doing process engineering for fluorochemical processing. When I went to grad school, I was captivated by the idea that you could do computational science for your PhD and the computer could be the thing doing the experiments for you. My PhD was in the area of computational chemistry and computational modeling of chemical reaction networks—everything from the physics of how molecules work to graph theory representation of large networks of chemicals. Tons of overlap with computer science when you get to that side of the spectrum.

The next big pivot for me was into machine learning and AI, which I still use heavily in my research. Back in the early 2010s, lots of grad students at the University of Washington were coming to me and saying, "There's this job called Data Scientist, but they won't give me an interview even though I'm pretty good at computers and I know statistics." That was an opportunity to reframe and update the curriculum for graduate students in chemical engineering and material science. We were one of the first departments

Provost Pfaendtner chats with students during the annual POP Back event on Centennial Campus (photo by Adam Jennings).

to build out a comprehensive machine learning and data-driven curriculum for chemical science, material science, and chemical engineering. I've been actively involved in my research group in the use of data-driven models—deep-learning models at first and now generative models. One of my current PhD students is working on Adaptive Reasoning Models right now for chemical and molecular discovery.

Sherard Griffin: How do you see this accelerating work in the computational sciences or chemical engineering? What is it allowing people to do that they couldn't do before?

Jim Pfaendtner: That's what makes me so excited right now. At NC State, we have people in all the disciplines working at that interface of data-driven engineering, machine learning, and deep learning. We have autonomous and self-driving labs with incredible expertise. In biomedical engineering, we have exoskeletons trained by deep learning and AI models to enhance well-being and quality of life for people who might be paralyzed or healing. I could go on and on—some faculty member is going to hear me and say, "Hey, you didn't mention me," but every department has these incredible strengths right now. We are by far the largest college of engineering and computer science in the state of North Carolina, and it brings together a massive amount of talent.

Sherard Griffin: I want to ask about your background, but first, tell me about your last name, Pfaendtner. What's the history?

Jim Pfaendtner: The family name is German. We looked up its origin and learned the Pfaendtner was like the repo man in the Middle Ages. If you got a payday loan and you didn't pay it back, the Pfaendtner came and broke your legs. Still in Germany today, if you pay a bottle deposit, it's called the "pfand."

Sherard Griffin: How did you get into engineering and computing?

Jim Pfaendtner: I grew up loving science and technology—I was obsessed with computers. Then I went to an engineering summer camp at Georgia Tech and I never looked back—I fell in love with Georgia Tech and went there as an undergrad. I moved to NC State three years ago from the University of Washington, where I started my professional academic career. Just like industry technical companies, universities have technical tracks and leadership tracks. I switched to the administrative leadership track about eight years ago, and I have loved academic leadership and all the opportunities that come with it.

Sherard Griffin: What's that first computer you had? I think we all have fond memories of someone saying, "Hey, here's this cool box you need to start playing around with."

Jim Pfaendtner: My family came from humble means. We had an Atari, but you can't do much with it. Then my dad brought a desktop

PC home from work, and within a week I took the whole thing apart. I got a screwdriver, slid the big case off, took the processor out of the motherboard, laid it all out on the table, and I had no idea how to put it back together. I got in a lot of trouble.

Sherard Griffin: Of course. Here's what could be a controversial question for our readers. What was the operating system on that first machine?

Jim Pfaendtner: Oh, no question. It was MS-DOS, probably 4.0.

Sherard Griffin: Nice. What was your first thought going from DOS/Windows to Linux?

Jim Pfaendtner: It was hard to understand what it was at first. It was really a revelation. My uncle worked at NASA, and as soon as he realized I had a modem, he taught me how to log into a supercomputer he ran and how to use email. It was very much a novelty as a 10th grader to load Pico and send my uncle an email using Pine. In retrospect, he probably shouldn't have given me that account. I don't think it would pass cybersecurity training today to give your nephew five states away an account on a supercomputer, but it was the wild west back then.

TEACHING AI FOR THE NEXT GENERATION

Sherard Griffin: It feels like the wild west now, too. What are some of the challenges you're seeing on the academic side of AI?

Jim Pfaendtner: Big public mission-focused land-grant universities like NC State, we're built like tanks. We're built to produce results for the state and to be reliable institutions that serve. Yet at this moment, being agile is essential. Every August, we're welcoming about 1,850 brand-new freshmen into the College of Engineering. How are we teaching them to think about AI? We're trying to figure out the minimal set of ideas, concepts, and hands-on experiences college freshmen need to begin to think about the engineering design cycle. That's why we have a comprehensive freshman experience, and we've worked hard over the past two decades to build it because we know it leads to better outcomes for students and better engineers.

But because we built that big durable thing, we've got to figure out, where does the dose of AI come early on and then where do we keep redosing it to reinforce it? In higher education, you've got to learn the concepts of something, you've got to practice it in a structured and safe way, and then you've got to get some dirt under your fingernails and do some real projects. Of course, they're using AI all the time already, but it's essential that we guide them in appropriate ways to do it, and that we guide them to do it in an engineering-forward, computer-science-forward way, so the fundamentals are still there.

Sherard Griffin: I think that's exactly right. We do a lot of work with early talent—interns or newly hired engineers—and we're changing the tires as we're driving. One of the things we're seeing is, yes, you absolutely have to have the fundamentals, because if you're just telling AI what to do, but you don't understand how to debug, how to investigate whether you're exposing something from a security perspective,

How are we teaching brand-new freshmen to think about AI?

For people who do interdisciplinary work in physical and engineering sciences, open source means we publish our work first.

or how to ask the right questions, you'll be in trouble. If you just passed the course and didn't really pay attention, then man, by the time you figure out what AI is doing, it may be too late in your development process.

Jim Pfaendtner: 100%. You can't replace the time and space a college student has to learn fundamentals and critical thinking. Even biologically, we know you can't replace it because the brain is, for lack of a better term, hardening. Students start learning at a slower rate. Their ideas and concepts about problem solving become somewhat calcified over time. We take this seriously, and we're lucky to have so many faculty who care deeply about this and are eager to jump in. I've heard near zero complaints from faculty about the use of AI—just incredible enthusiasm. Everything from incorporating it into the curriculum to making their jobs easier.

Sherard Griffin: Speaking of faculty: how has coalescing around AI changed your approach to faculty, either on the training side or the recruitment side?

Jim Pfaendtner: You won't be surprised that the early-career faculty we're hiring right now—maybe it's their first faculty job, maybe they've been one or two years somewhere else—are coming with all kinds of ideas for AI already, so we don't have to do much other than get out of the way. But we have faculty at all stages of their career. Where do we create that space for faculty to upskill? Industry does this well. The value proposition in industry for employee retention and employee upskilling is really clear. Higher education is not known for being particularly effective at this. We're launching a program right now to create a faculty fellowship program so people can step away on their sabbatical with the explicit purpose of upskilling in AI. For example, we have a faculty member in civil and construction engineering who is going to learn how to use AI to support her research in water quality, and the other stories are all equally motivating. These individuals will come back with amazing ideas for new classes, new research projects, and new grants.

INDUSTRY PARTNERS AND OPEN SOURCE

Sherard Griffin: I'd be remiss not to mention the partnership between Red Hat and NC State—me being an NC State alum. I remember being a student on NC State's campus and Red Hat's headquarters was right down the hall from some of my classes. My AI class on Centennial Campus was my first foray into AI. What do industry partnerships mean for the College of Engineering? How can a company like Red Hat help with some of the goals you're trying to achieve?

Jim Pfaendtner: We say it loud and we say it often: we're a public land-grant institution. In the College of Engineering, that means we partner with companies, nonprofits, and government. In the company space, most of our graduates go work in the private sector. We have a responsibility to engage deeply with companies and it's a two-way street. Companies get access to the best talent, but they're also giving ideas to our faculty and administrators. That's one of the first things you and I talked about: you asked me, "How are you going to approach the agentic software engineering problem and

incorporate that into the classroom?" and I said, "Well, I'm going to approach that by talking to Red Hat more with you." If we don't do that, we're missing a huge opportunity. The coming years are going to see a deeper engagement of companies in this type of arrangement, especially given the way the federal portfolio is changing, and the companies that are first at the well are going to get a lot out of it. I'm excited to continue structuring those relationships and working with companies that want to partner with us, because it's going to be great for everybody.

Sherard Griffin: At Red Hat we do something called open source—I don't know if you've heard of it.

Jim Pfaendtner: Yeah, a little. <laughs>

Sherard Griffin: What does open source mean to you? How did you get experience with it personally, and then how does that apply to the College of Engineering.

Jim Pfaendtner: It was my first year in grad school, and I got a new computer. You give a new student a new computer, and they say, "Install RHEL." I was like, what's that? Northwestern had a site license for Red Hat Enterprise Linux, so that taught me the benefit of an open source concept where you licensed access to libraries and software. I built my own cluster and installed Rocks and the operating system on it when I was a grad student—24 nodes!

But especially for people who do interdisciplinary work in physical and engineering sciences, open source means we publish our work first. I have never been the lead on a patent as an academic, but I've been part of over $40 million of research grants as PI or Co-PI, and that has led to probably 150 peer-reviewed publications coming out of the Jim Pfaendtner research group.

In the knowledge economy, publications are the open source way we transmit information. That's changing a little bit with this rapid rate of production of new information with AI and autonomous science, so publications might look different, but federal funding agencies require things to be open source. They require publication and free access to many types of products that come out of grants. So my bent has always been that I'm going to disseminate knowledge first, and then if I get involved in technology transfer or commercialization, that's fine. I support that, and many important startups have come out of universities. The amount of tech transfer that occurs is a huge benefit of the federal investment in research. But alongside that we also have to have open source software, open source research in AI, and open source dissemination of knowledge from all of our labs.

The amount of tech transfer that occurs is a huge benefit of the federal investment in research.

Sherard Griffin: It used to be if you were doing something AI, you needed access to a supercomputer and you submitted a job—maybe it was running on Slurm—and it took a while and then you got some results back. What has open source done to AI to make it more accessible?

Jim Pfaendtner: The push to make models smaller, more computationally affordable, to use less energy, all of that, is creating downstream benefits on campus. A lot of what we do in teaching students doesn't need the latest or even the previous generation of quality. An engineering example is the drift away from MATLAB toward Python over the past 15 years. Many more engineering classes are now using Scientific Python, Python, NumPy, etc. because it's free and open source. I taught with MATLAB; I learned MATLAB as a student. When I went to work at 3M, I needed to solve a problem and I asked for a MATLAB license. They were like, "We're not buying that." It was an aha moment for me when I was a young process engineer. So when I was department chair at University of Washington in chemical engineering, we wholesale switched all software to open source. Every class solved chemical engineering problems using Python. It was a bit of an abrupt shift, but we got through it.

DON'T JUST TALK ABOUT AI—DO IT

Sherard Griffin: Your work gives you a view across various domains in the College of Engineering. What are some of the areas you're most excited about?

Jim Pfaendtner: I just came from Washington DC at a gathering of engineering deans, a policy forum. Every year there's an hourlong panel on AI, and it's an echo chamber saying, "We need AI, we need to

The thing I am most excited about in engineering is the way we are going to transform engineering design.

teach AI, we need to do AI." We need to stop saying "You have to do AI" and dig into, "What are you doing? How can we work together?"

The thing I am most excited about in engineering is the way we are going to transform engineering design. A specific example would be a structure design in the civil engineering department. The way structural design is conducted has been largely unchanged since we've started building: before we were writing things down, we intuited physical laws and learned how force is distributed, then we wrote down those mathematical equations after Newton and we began to design around that, and then in the past 20-30 years we've used finite elements and supercomputers to enhance the way we design structures.

But at the end, it's an acceleration of the same thing. We're solving mathematical equations based on physics-based models. Now, data-driven models are emerging. We are going to get new insights on structures simply by processing the data of all structures humans have ever created and how those structures work. And as we continue to add more sensors on things, that is going to rapidly accelerate. A Microsoft researcher in 2009 called this The Fourth Paradigm. The Fourth Paradigm is starting to touch many traditional engineering disciplines, and it is going to revolutionize things.

Sherard Griffin: That's a great example of one of the unsung-hero types of stories about how AI is helping people sift through massive amounts of data that have been collected over the course of 20-30 years. We used to have these big analytical engines, and you would have to know the exact question to ask the unstructured data to get value out of it. Now, we're getting to a point with AI where it's helping bridge that gap. I may not know exactly what to ask, but I'm able to get through all of this data, whether it's structured or unstructured, and it can infer what I mean from somebody or even recommend, hey, is this really what you want to interpret?

So, we talked about the exciting part, but what keeps you up at night? Why are you not sleeping when it comes to the future of AI?

Jim Pfaendtner: What concerns me is how we are going to resource universities to have access to the technology. The arms race in the acquisition of processors is a challenge. The good news is we don't need the current generation of processors, but to do some of the exciting things we're talking about, we do need new datacenters; we're going to need an older generation of liquid-cooled chips. I really appreciate Red Hat's perspective on parsimony—having small language models that do bespoke tasks with way less power usage. That's going to be transformational, not only to things like small wearables but also the way we educate.

Sherard Griffin: Absolutely. We see this even in the enterprise, with our customers' challenges getting access to hardware, whether it's the GPUs, or now there's all kinds of issues with memory. Everyone wants access to

this hardware. Everyone wants access to these datacenters, but there's not enough to go around. How does something like the availability of hardware impact your researchers with some of the more advanced research they want to do? When I think about academia, I always think, okay, they're at the forefront of a lot of this, but what happens in a world where academia can't get access to the hardware because there's a hyperscaler that's requested all of the GPUs? If this doesn't correct itself, what could the impact to academics be?

Jim Pfaendtner: It's a good question, and I'm a little more optimistic here, because much of what those hyperscalers are doing is deploying an existing product for a million, 100 million, just pick your power of 10. But the original design and creation of that product doesn't necessarily need as much compute resource. Just the other day, our IT director for the college of engineering was chatting with me about a potential corporate donation of some older hardware, and he said, "I don't want to waste the time installing it if no one's going to use it." And I'll tell you, if academics are good at anything, it's using every table scrap available.

Sherard Griffin: So you look at AI, you look at what's transpired over the past few years. Researchers and faculty and students are a key part of this revolution. A lot of open source projects as well as some of the technologies and algorithms come out of research papers. And the time it takes to go from a research paper to deliverable software has shrunk dramatically. It used to be years. Now it's months. With how fast this is moving and how much this is being driven by researchers, what do you want NC State's contributions to be when we talk about AI a few years from now?

Provost Pfaendtner addresses the 2025 inaugural summit of the Bezos Center for Sustainable Protein, an initiative to bring innovative science and technology to bear on global food systems (photo by Adam Jennings).

Jim Pfaendtner: So those things have an original home or idea in academia, and they got transferred quickly because of best practices for software. As we develop the next generation of robotics operating systems for an exoskeleton or a drone or wireless whatever, I want NC State faculty to be doing their part to make it reproducible and maximize the velocity of dissemination. That involves making sure the software is attached to publications, it's documented well, and it's reproducible. In my own group, we strive for all of our research to be reproducible. And since it's computational, that should be achievable if we share all of our input and instruction.

We really want to emphasize our role in the dissemination of knowledge. It is essential that we do that at that intersection of computation and the physical world. I think it is an area where NC State could be really well-known, given that we've got a world-class computer science department embedded in a world-class flagship college of engineering. That's rare. It doesn't always happen, but I think the sky's the limit on how we're going to be able to to work together to achieve some of those goals.

Sherard Griffin: That's awesome, and a perfect place to end. Thank you.

Red Hat

RESEARCH QUARTERLY

VOLUME 8:1

Feature

CRANE: teaching code models to think without breaking their tools

Can we enhance AI reasoning without sacrificing the reliability of coding tools? The CRANE method proves it's possible.

by Mingzhi Zhu, Stacy Patterson, Michele Merler, and Raju Pavuluri

A stronger reasoning model is not automatically a better coding agent. For many AI systems, the standard approach is to take a model that can reason longer, plan more carefully, and recover from mistakes, then place it inside a more complex workflow. In agentic coding, however, the workflow itself is part of the task. A coding agent must not only solve a programming task, it must read files, follow repository conventions, call tools in the right manner, obey formatting protocols, run tests, interpret failures, and stop when the work is done. If the agent calls a tool too early, ignores instructions, loops after a failed command, or keeps thinking instead of testing a fix, the extra reasoning budget can translate into worse performance, in addition to the increased cost deriving from an expanded token count.

This is the motivation behind CRANE (Constrained Reasoning Injection for Code Agents via Nullspace Editing), a model-merging approach that explores a practical question: *can we give an Instruct model some of the planning and recovery strengths from a Thinking model while preserving the tool-use behavior that makes the Instruct model dependable?* Rather than assuming that a single checkpoint must be trained to do everything well, CRANE takes a compositional view of open agentic systems: different checkpoints can encode different strengths, and those strengths can be combined without retraining the model or rebuilding the agent interface. CRANE treats the difference between paired Thinking and Instruct checkpoints as a pool of candidate reasoning edits. It then filters, scores, and projects those edits so that useful reasoning behavior is transferred while tool protocols remain stable. The large-scale evaluation and ablation experiments underlying CRANE were supported in part by resources provided through the NAIRR pilot and the AI Alliance/MOC/IBM Research collaboration.

THE CRANE METHOD

Modern open model families increasingly provide multiple checkpoints tuned for different behaviors from a common set of base weights. In a paired setup, an Instruct checkpoint is often tuned

to be concise, protocol-following, and responsive to direct user or tool instructions. A Thinking checkpoint is tuned to spend more computation on planning, intermediate reasoning, and difficult problem solving.

On agentic coding tasks, the Thinking checkpoint may plan more, but it can also overthink. The Instruct checkpoint may obey tools more cleanly, but it can be too shallow when the task requires recovery or multi-step diagnosis (see **Figure 1**). Rather than choosing one option over the other, the goal is to transfer the parts of Thinking that help with agentic work while preserving the parts of Instruct that make a tool-using system usable.

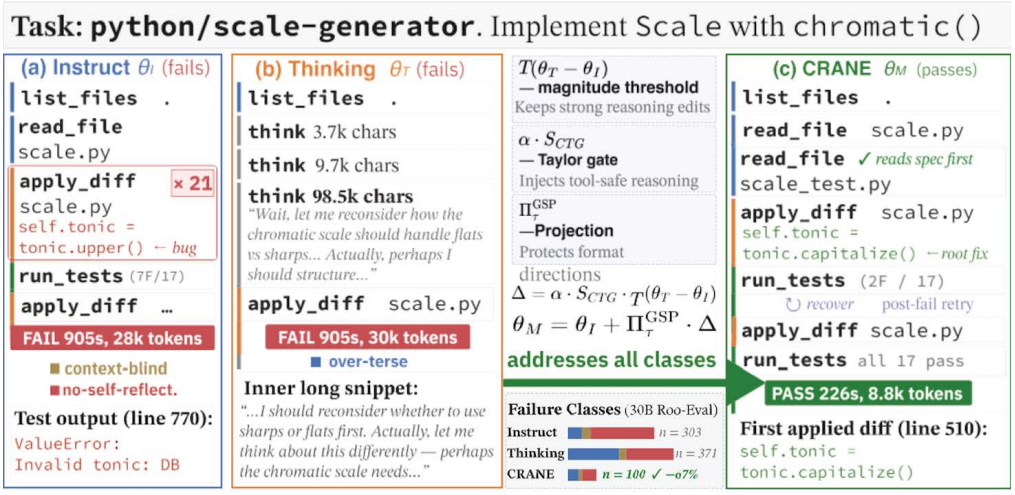


Figure 1. *Illustration of the endpoint trade-off: the Instruct model acts quickly but fails because it acts before checking the requirements. The Thinking model plans carefully but gets stuck in long reasoning loops without fixing the issue. CRANE succeeds by combining the best of both. The inset summarizes the common types of errors observed in failed attempts.*

From endpoint substitution to behavioral editing

The most direct way to use a Thinking model in a coding agent is endpoint substitution. This is simple, but it assumes that reasoning quality is the dominant variable. In practice, that assumption breaks down. Another common approach is model merging. If two checkpoints share an architecture and come from the same model family, we can combine their weights, for example, by interpolating between the Instruct and Thinking parameters. Such symmetric merging treats all parameter differences as if they are equally valuable. For agentic coding, this is not the case. Some differences may improve planning. Others may lead to longer deliberation that increases token cost without improving execution, or they may disturb formatting, tool-calling conventions, or instruction-following behavior. The challenge is to distinguish between these helpful and harmful differences.

CRANE starts with the delta between the Instruct and Thinking models

$$\delta = \theta_{\text{thinking}} - \theta_{\text{instruct}}$$

where θ_{thinking} and θ_{instruct} are the parameters of the Thinking and Instruct checkpoints, respectively. CRANE treats the δ as a pool of possible edits to the Instruct model. The merge is guided by three behavioral targets:

1. Reasoning transfer: move the Instruct model toward useful Thinking-generated behavior on code-reasoning prompts, especially planning, context integration, and recovery after failed attempts.

2. Agent-behavior preservation: keep the Instruct model's tool-use policy, including when to inspect files, when to edit, when to run tests, how to respond to observations, and when to stop.

3. Format preservation: protect tool-call formatting and protocol tokens that make tool calls valid, including chat-template markers, tool delimiters, JSON/schema structure, braces, and schema-critical keys.

The CRANE pipeline

CRANE is designed around three questions.

1. Which parts of the delta are large enough to impact model behavior?

2. Which edit directions help reasoning transfer without hurting tool-use behavior?

3. Which updates should be suppressed because they interfere with protocol-sensitive subspaces?

To address these questions, CRANE employs three stages:
1. Magnitude Thresholding,

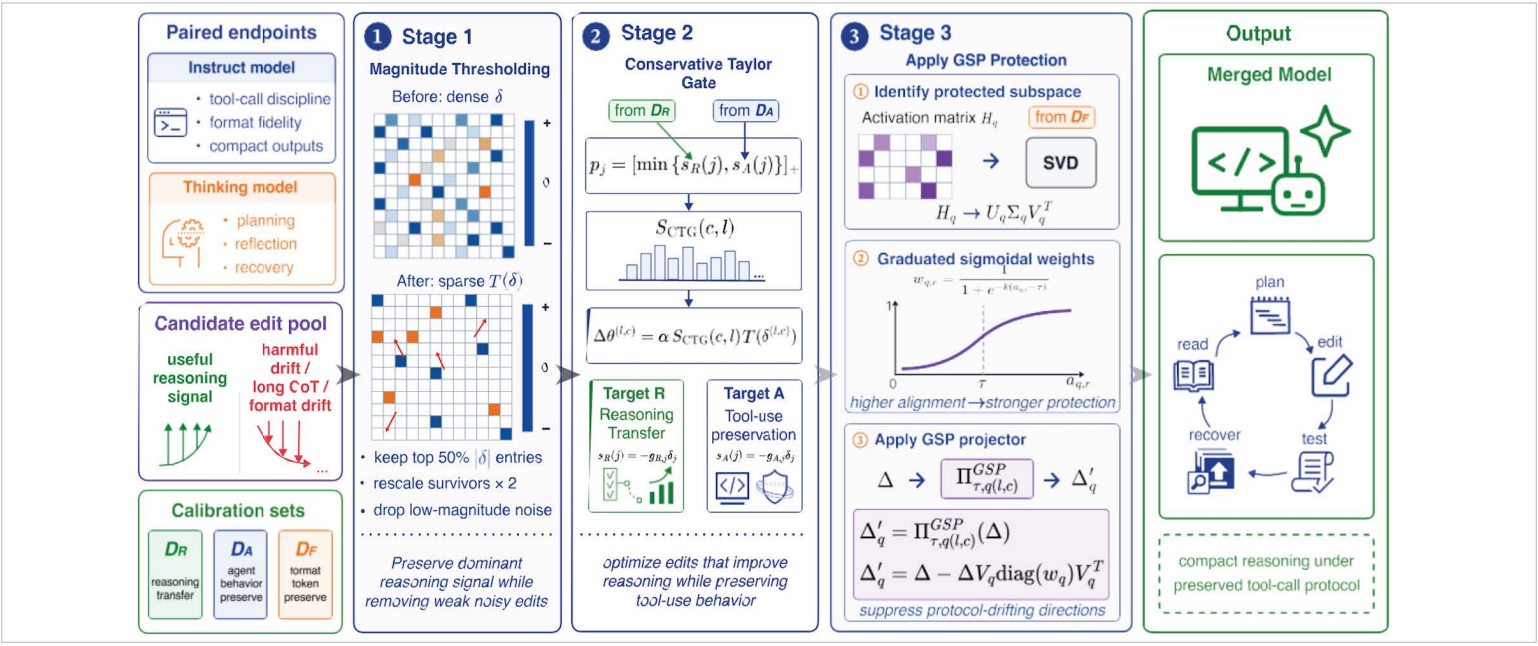


Figure 2. *The CRANE pipeline process. It functions in three steps: sparsifying data, filtering changes to prefer those that benefit both reasoning and tool usage, and protecting specific formatting rules so the model can continue to communicate with tools correctly.*

2. Conservative Taylor Gate, and
3. Graduated Sigmoidal Projection

$$\theta^{(l,c)}_{\text{merged}} = \theta^{(l,c)}_{\text{instruct}} + \underbrace{\Pi^{\text{GSP}}_{\tau,q(l,c)}}_{\text{stage 3}}\Big(\alpha \cdot \underbrace{S_{\text{CTG}}(c,l)}_{\text{stage 2}} \cdot \underbrace{T(\delta^{(l,c)})}_{\text{stage 1}}\Big)$$

where l indexes the transformer layer and c indexes the parameter component (see **Figure 2**).

In the first stage, *Magnitude Thresholding*, CRANE sparsifies the raw Thinking-Instruct difference by keeping only the larger-magnitude coordinates and dropping the smaller ones. A small weight difference may reflect incidental endpoint drift rather than useful reasoning transfer, but it can still perturb the model interface. This stage acts as a denoising pass. By eliminating lower-magnitude coordinates, CRANE reduces the set of candidate edits for later stages to those likely to make a significant impact on model behavior.

The second stage evaluates candidate edits against both reasoning and tool-use objectives using an algorithm called the *Conservative Taylor Gate* (CTG). It uses two small calibration sets, one containing code-reasoning prompts whose target continuations are generated by the Thinking checkpoint, and one containing tool-use repair prompts whose targets are generated by the Instruct checkpoint. These are not large, supervised fine-tuning datasets, but rather small signal sets used to estimate, at the Instruct endpoint, whether moving along a particular Thinking-side direction is locally helpful. The CTG scores each parameter according to whether the proposed edit reduces both reasoning-transfer loss and the agent-behavior preservation loss to first order. If a parameter edit helps both reasoning and tool-use, it gets a positive score; edits that harm both receive a negative score. Edits helpful for one but not the other get zero. Scores are then aggregated by layer and component, producing blockwise injection strengths rather than one global merge strength for the whole model. CRANE gives each layer-component block its own measured contribution.

The third stage protects the low-level protocol with *Graduated Sigmoidal Projection*, a filtering technique used in AI parameter merging. Even if an update improves reasoning and tool-use behavior in aggregate, it can still

damage the local computation around format-critical positions: JSON keys, delimiters, braces, tool-call syntax, or chat-template tokens. A malformed tool call can disrupt an otherwise good plan. CRANE collects Instruct activations around these format-critical tokens and builds a protected subspace for each relevant tensor. If an edit would significantly change the Instruct model's computation on these protected activations, that part of the update should be suppressed. CRANE tries to move the model toward better reasoning while keeping the update close to null on the activation directions that carry the agent protocol. It achieves this using a smooth spectral projection rather than a hard cutoff, so high-risk directions are removed nearly completely, low-risk directions pass through, and boundary cases are attenuated gradually.

RESULTS

We evaluated CRANE on three tool-using coding settings: Roo-Eval for in-IDE programming tasks, SWE-bench-Verified for repository-level issue resolution, and Terminal-Bench v2 for long-horizon command-line workflows. We used the Qwen3 series of LLMs and tested paired Qwen3 checkpoints at two scales: Qwen3-30B-A3B and Qwen3-Next-80B-A3B. The comparison included the original Instruct and Thinking endpoints plus several standard merge baselines, including Task Arithmetic, TIES, SLERP, AIM, LEWIS, and RAIN-Merging.

Figure 3 compares task success against TTC, our total token-cost proxy. TTC combines non-cached input tokens, cached-prefix tokens, and output tokens with weights that reflect their relative serving cost: TTC = input + 0.1 × cached input + 5 × output.

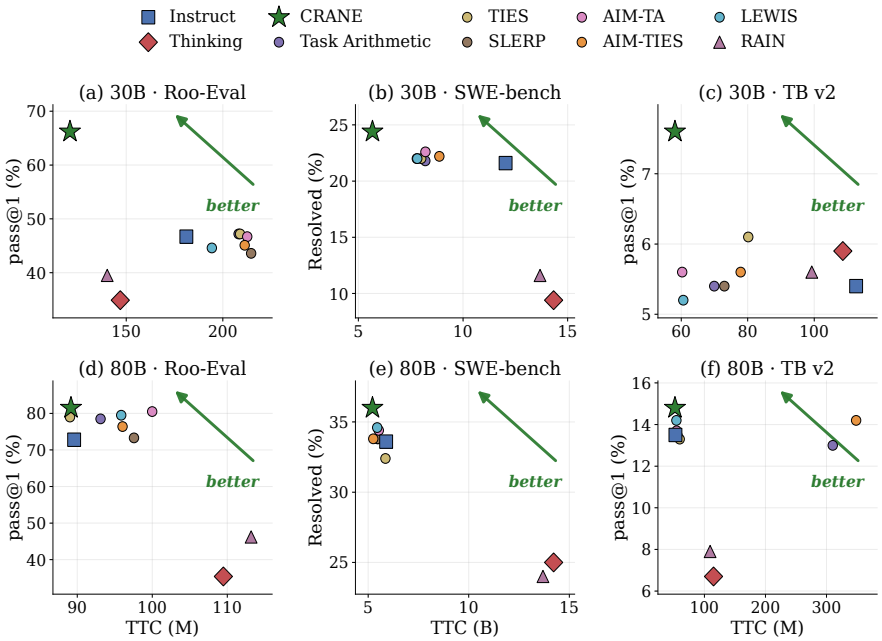


Figure 3. *TTC vs. pass-rate, three benchmarks × two scales. (a–c) Qwen3-30B-A3B on Roo-Eval, SWE-bench-Verified, Terminal-Bench v2; (d–f) Qwen3-Next-80B-A3B on the same three*

Qwen3-30B-A3B	Roo-Eval Pass@3	SWE-Verified Resolved	Terminal Bench V2 Pass@5
Instruct (ref)	64.1%	21.6%	10.1%
Thinking (ref)	52.8%	9.4%	13.5%
CRANE	83.1%	24.4%	17.9%

Table 1. *Selected pass rates among the three tool-using code-agent settings*

On Roo-Eval, CRANE improved 30B pass@1 from 46.7% for Instruct and 34.9% for Thinking to 66.2%. At 80B, it improved pass@1 from 72.8% for Instruct to 81.5%. These gains did not come from simply spending more tokens. At 30B, CRANE reduced the total token-cost proxy by 60.2 million relative to Instruct while improving pass@1, pass@3, and pass-all.

On SWE-bench-Verified, CRANE resolved 122 of 500 tasks at 30B, compared with 108 for Instruct and 47 for Thinking. At 80B, it resolved 180 of 500, compared with 168 for Instruct and 125 for Thinking. It also used a lower aggregate token-cost proxy than the Instruct reference at both scales.

Terminal-Bench v2 offered a different stress test: long shell workflows in cloud sandboxes. CRANE reached the strongest pass@1 and pass@5 results at both scales. At 30B, it achieved 7.6% pass@1 and 17.9% pass@5 (reflected in **Table 1**), while finishing nearly two hours faster than the Instruct reference. At 80B,

it reached 14.8% pass@1 and 30.3% pass@5, again with compact output relative to the Thinking-style rows.

The same pattern appeared across benchmarks. Standard merge baselines sometimes improved over an endpoint, especially at 80B, but their gains varied by benchmark. RAIN-Merging, which is designed for the reverse direction of preserving a thinking format while importing instruction-following ability, often retains thinking-like verbosity. CRANE was more reliable in the direction code agents need: importing reasoning into an Instruct-style agent while preserving the needed agent behavior.

WHY THIS MATTERS FOR OPEN SOURCE AI SYSTEMS

A practical advantage of CRANE is that it is training-free. It requires no supervised fine-tuning or reinforcement learning, and no rebuilding of the agent interface. For open source model families with paired checkpoints, this weight-space editing can simplify deployment. One can start with the model that interacts reliably with tools and selectively inject reasoning capabilities from the paired checkpoint.

CRANE has its limitations too. It assumes the existence of paired checkpoints with complementary strengths: a Thinking checkpoint providing useful reasoning and an Instruct checkpoint that handles tools interaction correctly. Calibration traces must cover the range of tool interactions expected during deployment. If tools, schemas, or stopping rules change substantially, the protected behavior may need recalibration. Finally, building the format-subspace projectors requires forward passes through the Instruct model, which can be costly for very large checkpoints.

Overall, the results suggest a practical direction for open agentic systems. Rather than training a single checkpoint to do everything well, we can combine checkpoints with different strengths. CRANE shows that it is possible to transfer reasoning improvements from Thinking models while preserving the reliable tool-use patterns of Instruct models. This approach is particularly relevant in software-engineering tasks, where reliable tool use often matters more than incremental gains in raw capability.

Acknowledgement

The authors acknowledge the National Artificial Intelligence Research Resource (NAIRR) Pilot, the AI Alliance and the Mass Open Cloud for contributing to this research result.

About the Authors

Mingzhi Zhu
is a second-year PhD student in Computer Science at Rensselaer Polytechnic Institute, advised by Professor Stacy Patterson. His research focuses on parameter-space model merging techniques for code agents.

Stacy Patterson
is an Associate Professor and the Associate Head of the Department of Computer Science at Rensselaer Polytechnic Institute. Her research focuses on distributed computing systems, machine learning, and data privacy.

Michele Merler
is a Principal Research Scientist at IBM's T. J. Watson Research Lab in New York. His work focuses on multimodal modeling in AI4code, computer vision, digital medicine, and NLP. He is a Senior Member of the IEEE.

Raju Pavuluri
is a Senior Technical Staff Member and Research Manager at IBM, leading research in AI4Code, software engineering, and application modernization. His work spans AI-driven development, agentic systems, code analysis, and automated testing, bridging cutting-edge research and large-scale enterprise software systems.

What is Red Hat developing

NEXT ?

Learn more at

next.redhat.com

Red Hat

RESEARCH QUARTERLY

VOLUME 8:1

Feature

Beyond the leaderboard: rethinking time-series foundation models

As zero-shot performance gains flatten, researchers are rethinking the field: separating architectural progress from data-driven gains, and introducing a language-driven paradigm for TSFMs.

by Yunshi Wen, Wesley M. Gifford, Chandra Reddy, Lam M. Nguyen, Jayant Kalagnanam, and Anak Agung Julius

Ask a power grid operator, a hospital administrator, or a logistics planner what keeps them up at night, and somewhere on the list will be a forecasting problem. How much electricity will customers draw next Tuesday? How many patients will arrive in the emergency department this weekend? How many containers will move through the port next month? For decades, these forecasts have been built using custom statistical models for each application. Each requires its own data, its own tuning, its own expert.

Over the past few years, the machine learning community has been chasing a more ambitious idea: what if a single model, trained once on a vast and diverse collection of historical time series, could forecast *anything*, including data from domains that it has never seen during training? This is the premise of a *time-series foundation model* (TSFM). TSFM borrows ideas from large language models: train the models on enough data of the right kind and the models will pick up general patterns, allowing them to produce accurate forecasts in new domains without retraining.

THE CHALLENGE

The standard test of this capability is called *zero-shot forecasting*: the model is shown a stretch of history from a dataset it has never encountered during training and asked to predict what comes next without fine-tuning or domain-specific adjustment. Performance is scored against ground truth future values using some standardized metrics, such as mean absolute scaled error (MASE) for point predictions and continuous ranked probability score (CRPS) for the model's full forecast distribution. The GIFT-Eval benchmark, which aggregates 97 test cases across seven domains, has become the field's de facto leaderboard.

That leaderboard has filled up quickly with models such as Chronos and Chronos-2 from Amazon, TimesFM from Google, Moirai from Salesforce,

and others. Each new release claims some architectural innovation that leads to stronger performance. The pace of progress is real. But it has produced a maturing field with two problems that are harder to see while the leaderboard is still moving.

The first problem is identifying the main contributors to the performance gain. When one model outperforms another on GIFT-Eval, the natural assumption is that the new architecture is responsible. Yet each team trains its model on a different mix of data, with different recipes, often using proprietary synthetic data generation pipelines that are described only in broad strokes. When pretraining corpus, training protocol, and architecture all change together, it becomes impossible to say which one is mostly responsible for the performance gain. The community has been comparing combined systems but interpreting the comparisons as if they isolated architectural progress.

The second is a capability problem, and it becomes more pressing as accuracy on the leaderboard saturates. The standard TSFM interface is narrow: a window of history goes in, a forecast distribution comes out. There is no clean channel through which a human, or another automated system, can guide the prediction using external knowledge or contextual reasoning. As leaderboard differences shrink to fractions of a percent, the more useful question is shifting from how to forecast slightly more accurately to accomplishing what these models cannot yet do at all.

TIME-SERIES DATA AGENT

The work described in this article emerged from a project at Rensselaer Polytechnic Institute, conducted jointly with collaborators at IBM Research and supported by the National AI Research Resource (NAIRR) Pilot and the Mass Open Cloud. The broader project aims to develop an agentic system, the Time-Series Data Agent, that couples a time-series foundation model with a large language model to enable language-driven forecasting, analysis, and control. Building that system required progress on both problems above. Before committing the project to a particular foundation-model backbone, the team needed to know which TSFM design choices actually matter and which are incidental. And before that backbone could be coupled meaningfully to a language model, the team needed a representation through which the two systems could interact. Two parallel lines of work addressed these questions. The first asked how much of recent TSFM progress is really attributable to architecture. The second asked what new capabilities become possible if the representation of time series inside the model is itself reconsidered. The findings from each, taken together, map a position for the field and a path for the larger project.

Architecture vs. data

To address the first question, the team built a deliberately unremarkable forecasting model around a standard encoder-only Transformer backbone, based on the PatchTST architecture developed earlier by the IBM collaborators. There were no time-series-specific modifications to the attention mechanism and no novel components in the backbone itself. The surrounding pieces—patch-based tokenization of the input sequence, a normalization scheme

When pretraining corpus, training protocol, and architecture all change together, it becomes impossible to say which one is mostly responsible for the performance gain.

that handles missing values, and a quantile-based output head for probabilistic predictions—were all assembled from techniques already established in the literature. The model was paired with a carefully documented training recipe built entirely from publicly available data, then compared against the specialized models at the top of the leaderboard.

The team's generic transformer-based model, PatchTST-FM, delivered top performance. It matched or exceeded the results of leading specialized TSFMs on both the GIFT-Eval and TIME benchmarks. Notably, among the top zero-shot performers on the GIFT-Eval leaderboard—excluding those with training data leakage—PatchTST-FM outperformed models like FlowState (IBM Research), TimesFM-2.5 (Google Research), TiRex (NXAI), and Chronos-2-Synth (Amazon) by a small margin.

Three factors contributed to the performance gain: the pretraining data, the masking strategy during pretraining, and the context-window length. The cumulative message is that a generic Transformer trained well on the right data, with contiguous masking and a long context window, is sufficient to compete with, and even beat, TSFMs that introduce specialized architectural components. The corollary is uncomfortable: many of the architectural innovations claimed over the past two years are likely carrying credit that belongs to differences in data and training. The team released their model checkpoints and full training pipeline as a public baseline, in part to make this kind of confounding harder to introduce in future work.

Single- vs. multi-scale

The findings above sharpened the second question. If the frontier of high-performance models is becoming densely packed, the productive question is no longer how to squeeze out a small gain of performance, but how to engineer new functionalities. In particular, the typical TSFMs currently offer no clean and practical method of intervention. A continuous patch architecture exposes only raw numerical values, which is the wrong level of abstraction for either human or machine reasoning to operate at. To couple a TSFM with a language model in any meaningful way, the representation itself needs to change.

Many of the architectural innovations claimed over the past two years are likely carrying credit that belongs to differences in data and training.

The second line of work, an architecture named ASTEP, for Abstract Semantic Tokenizer-Predictor, can facilitate intervention. The central idea is to translate the series into a sequence of discrete tokens drawn from a learned vocabulary of recurring temporal "shapes" and to use that vocabulary to describe time-series data across multiple scales. Coarse tokens describe broad structures, such as a gradual rise or a long plateau. Finer tokens describe the smaller fluctuations layered on top. The series is encoded as a coarse-to-fine hierarchy, in a manner loosely analogous to how text is encoded as sentences, words, and sub-word pieces. What is interesting about ASTEP is not discrete tokenization itself, which has been explored before, but the multi-scale residual structure.

Ablations show that a single-scale tokenizer operating only at patch resolution performs measurably worse than the multi-scale version. The hierarchy is doing real work. The plausible reason is that time series carry meaningful structure at many resolutions at once; for example, a daily oscillation rides on a weekly cycle, which rides on a seasonal trend, which rides on a longer-term trend, and so on. A flat tokenization forces the model to recover this hierarchy implicitly. A multi-scale tokenization makes it explicit in the representation, which has the further effect of making the model's internal reasoning legible in a way continuous-patch models are not: a coarse token corresponds to an interpretable structural decision, not an opaque vector.

RESULTS

The accuracy result is competitive. Even without intervention, ASTEP's large variant model achieves the best performance in zero-shot GIFT-Eval benchmarking, beating PatchTST-FM. Further, because the model represents forecasting as a hierarchy of tokens, a user or an external system can specify the coarse-level tokens directly and let the model fill in the fine detail consistent with that sketch. The team calls this *concept-level intervention*. Empirically, it produces

little benefit at short horizons, where there is no coarse structure left to specify. However, at long horizons where unconstrained forecasts diverge and uncertainty grows, high-level intervention translates into substantial gains in accuracy.

This is the piece that completes the picture for the larger project. A language model that has understood a user's request, or that has reasoned through a planning problem, has no clean mechanism for pushing its conclusions into a conventional TSFM beyond rewriting the input or post-editing the output. ASTEP's token hierarchy provides one. The language model can specify the coarse tokens, e.g., "expect a slow rise, then a plateau." ASTEP can produce the calibrated, fine-grained probabilistic forecast consistent with that intervention.

Taken together, the two lines of work map a position for the field. The first sets the floor: with a careful, public training recipe, even a generic architecture is competitive with the best-performing specialized models on leaderboards, and many recent leaderboard movements may not reflect genuine architectural progress. The second raises the ceiling: discrete multi-scale tokenization delivers comparable accuracy while adding a new axis of capability, controllable forecasting, that standard designs cannot support. For the sponsors of the project, the value is on both fronts. A transparent baseline lowers the cost of deploying a strong forecasting system in any domain. A foundation model that can be steered by language opens new patterns of use in planning, monitoring, and decision support, which is the eventual objective of Time-Series Data Agents.

About the Authors

Yunshi Wen
is a PhD student at Rensselaer Polytechnic Institute in the Department of Electrical, Computer, and Systems Engineering interested in interpretable machine learning, neuro-symbolic learning, and their applications in time-series analysis, robotics, and control.

Wesley M. Gifford, PhD
is Senior Technical Staff Member and Manager, IBM, leading efforts on time-series foundation model architectures and applications.

Chandra Reddy, PhD
is a researcher at IBM specializing in time-series foundation models, optimization, and their applications in industry.

Lam M. Nguyen, PhD
is a Staff Research Scientist at IBM Research.

Jayant Kalagnanam, PhD
is a Director, Algorithms & Applications, IBM, with a background in AI Optimization focused on algorithm discovery.

Anak Agung Julius
is a Professor in the Department of Electrical, Computer, and Systems Engineering at Rensselaer Polytechnic Institute.

Red Hat

RESEARCH QUARTERLY

VOLUME 8:1

Feature

Kubernetes meets GenAI: evaluating performance for AI inference workloads

As GenAI inference becomes a dominant workload, researchers are evolving new Kubernetes-native projects to address bottlenecks, delivering the scalability and efficiency AI workflows require.

by Sai Sindhur Malleni, Raúl Sevilla Canavate, Aleksei Vasilevskii, José Castillo Lema, and André Bauer

The rapid rise of generative AI has fundamentally reshaped cloud computing. From sophisticated training pipelines to high-throughput, real-time inference, modern AI workloads demand dynamic access to expensive, heterogeneous resources, particularly GPUs and specialized accelerators. Kubernetes, already the de facto standard for container orchestration, is increasingly being asked to manage these demanding workloads. But can it keep up? This question is particularly pressing as organizations—from research institutions sharing GPU clusters across teams to sovereign AI initiatives building domestic inference capacity—seek open source, vendor-neutral solutions to maximize the value of scarce accelerator resources.

In our new paper, "Evaluating Kubernetes performance for GenAI inference: from automatic speech recognition to LLM summarization," presented at the 17th ACM/SPEC International Conference on Performance Engineering (ICPE '26) in Florence, Italy, we set out to answer that question. The paper received the Best Industry Paper Award at the conference.

Building on our previous work on multicluster networking performance in Kubernetes (presented at ICPE '25), this study shifts the focus to a different frontier: how well the Kubernetes ecosystem can support the unique demands of generative AI inference. While traditional Kubernetes scheduling and resource management APIs work well for microservices, they were not originally designed for fine-grained GPU resource slicing, complex batch scheduling, or the specific traffic patterns of model serving endpoints.

THE CHALLENGE: AI WORKLOADS PUSH KUBERNETES BEYOND ITS COMFORT ZONE

AI inference workloads differ fundamentally from the microservices Kubernetes was originally designed to orchestrate. They require batch scheduling with priorities and preemption, dynamic partitioning of expensive GPU resources, and intelligent request routing that accounts for model-specific characteristics like key-value (KV) cache state. The standard Kubernetes primitives, while extensible, do not address these needs out of the box.

To bridge this gap, we evaluated three complementary Kubernetes-native projects that each address a distinct aspect of the AI inference pipeline:

- Kueue for job queuing and batch scheduling
- Dynamic Accelerator Slicer (DAS) for GPU resource partitioning
- Gateway API Inference Extension (GAIE) for intelligent inference routing

THREE KUBERNETES-NATIVE PROJECTS, ONE COHESIVE PLATFORM

Kueue is an open source project under the Kubernetes Scheduling Special Interest Group that extends native Kubernetes scheduling to support advanced batch and job management. It introduces a hierarchical queuing model with *LocalQueues* and *ClusterQueues*, enabling cluster operators to define resource boundaries, priority classes, and admission controls. This is particularly valuable for AI workloads where multiple teams share expensive GPU resources and need fair, predictable scheduling.

The Dynamic Accelerator Slicer (DAS) tackles GPU resource efficiency. In AI inference deployments, statically pre-slicing GPUs often leads to fragmentation and underutilization. DAS provides on-demand, just-in-time GPU slice allocation driven by actual workload requirements. Using NVIDIA MIG (Multi-Instance GPU) technology, it dynamically creates and destroys GPU partitions to match workload demands, improving utilization and reducing idle costs.

The Gateway API Inference Extension (GAIE) brings inference-aware networking to Kubernetes. Built on the standard Gateway API, it introduces domain-specific constructs for generative AI: model versioning, adaptive backend selection, and intelligent, prefix-cache-aware routing. Together with llm-d, a Kubernetes-native, high-performance distributed LLM inference framework, GAIE routes requests to the most suitable backend based on live model and accelerator metrics, including queue length, prefix cache hits, and LoRA adapter availability.

A REAL-WORLD USE CASE: FROM AUDIO TRANSCRIPTION TO SUMMARIES

To evaluate how these three components work together, we designed a multi-stage AI inference pipeline that reflects common industry usage patterns. The workflow processes a dataset of corporate earnings calls through two stages:

1. **Automatic Speech Recognition (ASR)**: Audio files are transcribed using OpenAI's Whisper models (medium and large variants), managed as batch jobs on Kubernetes.

2. **LLM Summarization:** The resulting transcripts are fed to a Qwen3-8B large language model for summarization, served through an OpenAI-compatible API endpoint.

We used the Earnings 22 dataset, comprising 125 financial earnings calls from various global companies, selecting a subset of 32 files for our experiments. All experiments ran on Red Hat OpenShift Service on AWS (ROSA) with a single GPU node featuring 8 NVIDIA A100 GPUs. We used kube-burner for workload orchestration and GuideLLM for benchmarking the LLM inference endpoint.

WHAT WE FOUND

Our evaluation yielded concrete, measurable improvements at each stage of the pipeline.

Scheduling with Kueue

Kueue's priority and preemption mechanisms reduced the total makespan—the time for all 32 transcription jobs to complete—by up to 15% compared to the baseline. More importantly, Kueue delivered deterministic, fair, and reproducible scheduling with negligible admission latency overhead (below 25 ms

Configuration	Makespan (Med, (Min–Max))	Queue Time (Med / P95)	Exec Time (Med)	Medium Jobs		Large Jobs	
				Queue (Med)	Exec (Med)	Queue (Med)	Exec (Med)
Pure Jobs (Baseline)	60.5 (59.1–60.5)	0.0 / 0.0	28.4	0.0	23.6	0.0	35.3
Kueue BestEffortFIFO	62.5 (62.4–65.8)	16.7 / 39.5	9.4	15.9	6.5	17.3	17.8
+ Priority	54.7 (52.2–56.1)	19.5 / 44.5	9.5	39.1	6.7	10.2	17.1
+ Preemption	51.1 (49.2–52.7)	25.0 / 41.9	8.9	36.7	6.4	3.7	15.6
2 ClusterQueues + Borrowing	55.0 (54.2–60.0)	13.7 / 32.4	8.7	10.2	6.7	25.1	18.0

Table 1. *Performance comparison of Kueue Configurations (96 jobs across 3 runs per configuration). All values are in minutes. Makespan median computed from 3 run-level makespans comprising 32 jobs per run; other medians computed from 96 jobs (32 jobs across 3 test runs) from each configuration.*

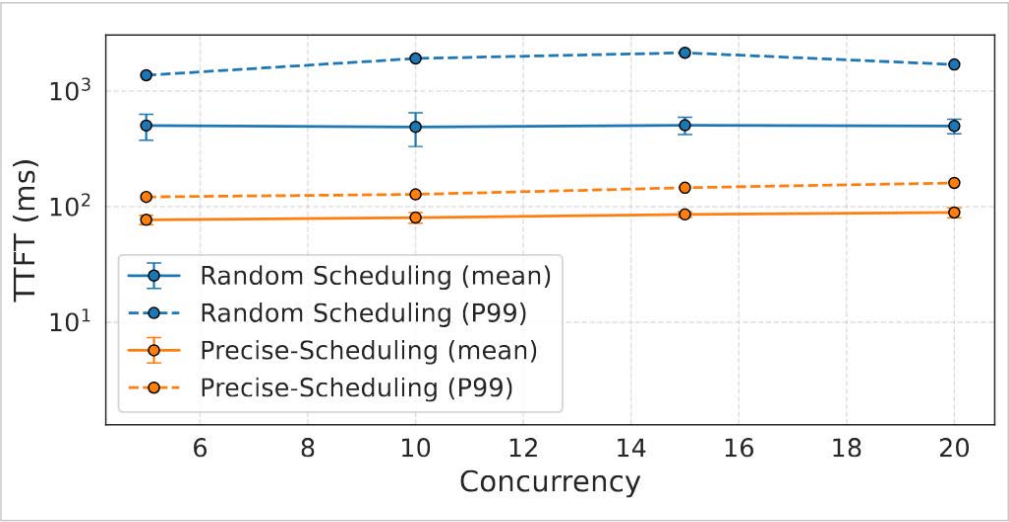


Figure 1. *Comparison of the Time to First Token (TTFT). Error bars indicate 95% confidence interval. Y-axis is depicted in log-scale.*

even at the 99th percentile). The configuration with two ClusterQueues and resource borrowing struck the best balance between fairness and utilization, allowing underutilized GPUs to be dynamically reallocated across queues. As shown in **Table 1**, while the baseline avoids queuing delays, its median execution time is considerably higher (28.4 vs. 8.7–9.5 minutes) because jobs contend for resources without admission control, while Kueue's managed admission trades modest queue wait times for significantly faster and more predictable execution.

GPU slicing with DAS

DAS increased the number of concurrently running transcription jobs from 8 (one per physical GPU) to 25 (using MIG slices), reducing mean job completion time by 36%—from 28 to 18 minutes. While individual jobs ran slightly slower on GPU slices compared to full GPUs, the dramatically increased parallelism more than compensated, resulting in significantly better overall throughput. GPU tensor core utilization and DRAM activity both tripled with DAS enabled.

Intelligent routing with llm-d

llm-d's precise-prefix-cache scheduling strategy, which leverages KV-cache events to drive inference requests to replicas with matching prefix tokens, delivered striking improvements over random scheduling thanks to an improved vLLM's KV-cache hit rate. The Time to First Token (TTFT) improved by approximately 6.25 times (from ~500 ms to ~0 ms), and end-to-end request latency dropped by up to 6 seconds at the 99th percentile. Precise-prefix-cache scheduling also exhibited significantly lower variability, suggesting better predictability under high concurrency (see **Figure 1**).

WHAT COMES NEXT

This work demonstrates that Kubernetes, when extended with emerging AI-oriented projects, can effectively serve as a unified substrate for both batch and online GenAI workloads. The complementary strengths of these

three components—Kueue for scheduling fairness, DAS for GPU efficiency, and GAIE for intelligent routing—form a cohesive, high-performance platform for demanding AI inference pipelines. These results carry broad implications beyond our specific benchmarks. As GPU resources remain scarce and expensive, the ability to efficiently partition and schedule accelerators through open source Kubernetes-native tooling is critical—whether for research groups sharing a cluster, enterprises optimizing inference costs, or sovereign AI programs building domestic AI infrastructure on transparent, auditable foundations.

Future work will focus on extending the evaluation to multimodal and streaming inference scenarios, integrating emerging components like gang scheduling and Dynamic Resource Allocation (DRA), and positioning this framework as a foundational blueprint for end-to-end AI workload orchestration on cloud-native infrastructure.

LEARN MORE

The full paper is available on the ACM Digital Library. All necessary files and configuration for replicating the test setup can be found on our GitHub repository.

About the Authors

Sai Sindhur Malleni is currently a Senior Engineering Manager at NVIDIA, where he leads initiatives focused on establishing Kubernetes as the common substrate for AI training and inference across the industry. His work centers on advancing open source technologies that make GPU infrastructure reliable, elastic, and seamless to operate at scale. Previously, he spent over a decade at Red Hat advancing OpenShift performance.

Raúl Sevilla Canavate is a Performance Engineer at Red Hat with a wide experience in Linux, automation, tooling, and performance analysis within the Kubernetes ecosystem for the last eight years. In recent years, he has been focused on pushing Kubernetes and OpenShift to its limits through CNCF's hosted kube-burner project.

Aleksei Vasilevskii is a Senior Systems Software Engineer at NVIDIA with broad experience optimizing the performance and reliability of high-traffic applications running on Kubernetes/OpenShift. His work centers on maximizing the throughput, resilience, and operational efficiency of Kubernetes-based environments tailored for GPU-accelerated workloads.

José Castillo Lema is a Software Engineer in the NVIDIA Technical Partnership Team at Red Hat Ecosystem Engineering. He has been designing and implementing IaaS/PaaS solutions, namely OpenStack and Kubernetes/OpenShift, and teaching postgraduate courses for the last ten years.

André Bauer is an Assistant Professor in the Department of Computer Science at the Illinois Institute of Technology. Previously he was a postdoctoral researcher in computer science at the University of Chicago and a researcher at Argonne National Laboratory. His research is on providing intuitive, efficient, and sustainable data science solutions across all disciplines and domains.

Red Hat

RESEARCH QUARTERLY

VOLUME 8:1

Feature

Monitoring latency within the end host network stack

Meet **netstacklat**, an open source eBPF tool developed with university researchers to provide critical visibility into host network latency and enable faster response times.

by Simon Sundberg, Anna Brunström, Simone Ferlin-Reiter,

Jesper Dangaard Brouer, and Toke Høiland-Jørgensen

Any application that communicates across the network (which today is most applications) is impacted by network latency, with high latency translating directly to worse application response times, which in turn leads to degraded quality of experience for users of the application. Network latency has typically been associated with long network routes or packets getting stuck in large queues at the bottleneck link. However, as higher network capacities have outpaced the development of end host resources, the bottleneck in the network is increasingly shifting towards the end host itself.

While researchers have spent significant effort investigating and identifying latency sources in the end host network stacks, there remains a lack of tooling to monitor this latency. Service providers therefore remain blind to the extent to which host latency impacts their service response times and are unable to determine whether latency issues stem from their network or the servers themselves. To enable continuous monitoring of host network latency in production systems, we have therefore developed **netstacklat**. Visibility provided by **netstacklat** allows service providers to pinpoint if host network stack latency is degrading response time and isolate network-level performance issues.

INDUSTRY-ACADEMIA COLLABORATION

The development of **netstacklat** is part of a long-term collaboration between Red Hat and Karlstad University. As part of the project "Building the next generation of programmable networking—powered by Linux," we have already looked at how we can use eBPF to continuously monitor network latency across internet connections. That work has been adopted by LibreQoS, a platform used by hundreds of internet service providers around the world, now monitoring the latency for millions of connections.

Building on this prior experience, we move our attention to monitoring the latency

within the end host itself. Our solution, **netstacklat**, once again employs eBPF to achieve low overhead while remaining fully compatible with the Linux network stack. As with prior work, we have upstreamed our code to the bpf-examples repository, making it freely available for others to use and build upon. We are working on adding **netstacklat** to libbpf-tools, allowing it to be packaged together with a wide range of powerful eBPF-based observability tools.

A slightly modified version of **netstacklat** has already been deployed fleetwide at Cloudflare, keeping track of how long it takes before requests start being processed. We are currently investigating different opportunities to integrate **netstacklat** with Red Hat's observability solutions (see "Boosting speed: Use eBPF and netstacklat to troubleshoot latency" on the Red Hat Developer blog). We will also present our paper "Waiting at the front door: Continuous monitoring of latency in the host network stack" at the highly regarded ACM Internet Measurement Conference (IMC '26) later this year.

THE KERNEL NETWORK STACK

The Linux kernel network stack represents incoming packets using a data structure called (for historical reasons) a *socket buffer*, or *SKB*. When a network device receives a data frame, the driver constructs a SKB to represent the packet (or attaches the frame data to an existing SKB) and submits the SKB to the kernel's network stack for processing.

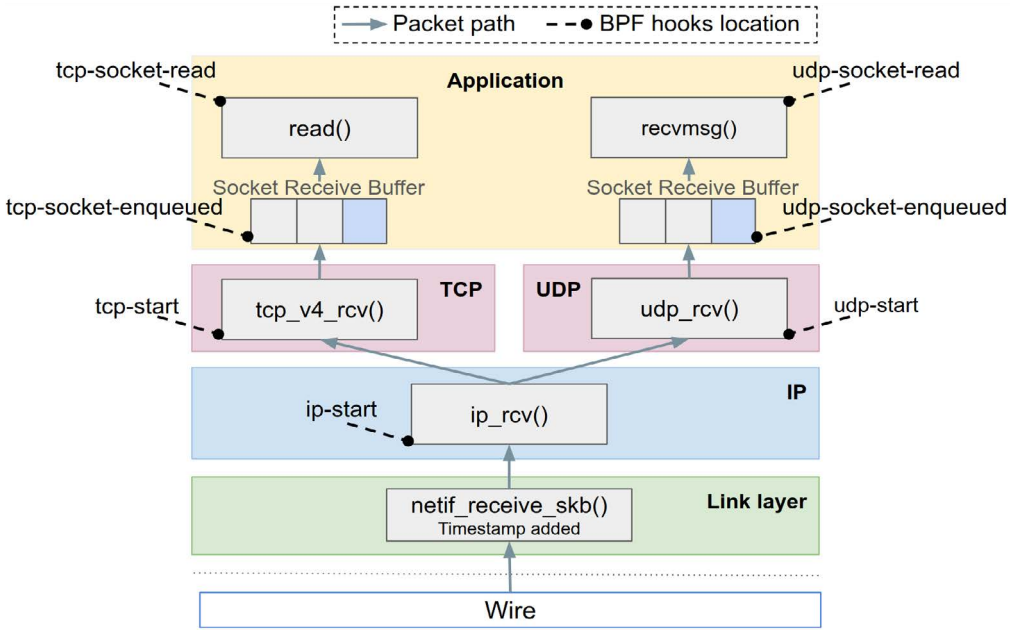


Figure 1. *The Linux ingress network stack and the* **netstacklat** *probe points*

The path the packet takes through the core networking stack is illustrated in **Figure 1**, which also shows at which points **netstacklat** measures the latency. The **netif_receive_skb()** function is the entry point of the network stack, called by the driver code. Protocol-specific functions then process each protocol layer of the packet. For the common TCP and UDP protocols, the packet is ultimately enqueued into the socket receive buffer of an application, where it is kept until the application reads the data.

DESIGN OF NETSTACKLAT

With **netstacklat**, we want to monitor how latency builds up as SKBs traverse the kernel network stack, from the start of packet processing until the application reads the data. To achieve this, we use eBPF programs to observe packets at several strategically selected points in the kernel and record the latency distribution at each point.

There already exist several open source tools that utilize eBPF to monitor SKBs in the network stack, such as **pwru** (developed by Cilium) and **retis** (developed by Red Hat engineers). What sets **netstacklat** apart is that instead of reporting the timestamp for each probe point to a user space agent, **netstacklat** calculates and aggregates the latency directly in the eBPF programs. As detailed in the next section, this drastically reduces the overhead for **netstacklat**, making it feasible to continuously monitor the network stack latency in production.

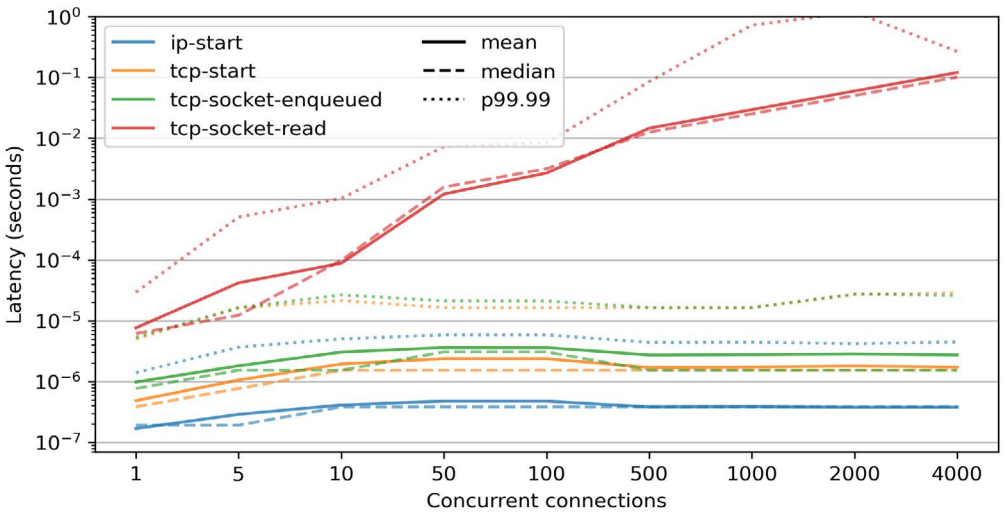


Figure 2. *Network stack latency as the number of concurrent connections increase*

To efficiently calculate the latency in the eBPF program, we make use of Linux's capability to timestamp incoming packets early in the network stack (SOF_TIMESTAMPING_RX_SOFTWARE), as shown in the bottom of Figure 1. By using the SKB receive timestamp as a reference point, we can determine the time an SKB has spent in the kernel network stack at any point by taking the difference between the current time and the SKB timestamp. We then aggregate the obtained latency value into a power-of-two histogram stored in an eBPF map. We have made sure the data format is compatible with **`ebpf_exporter`**, which makes it easy to expose the **`netstacklat`** latency data as Prometheus metrics.

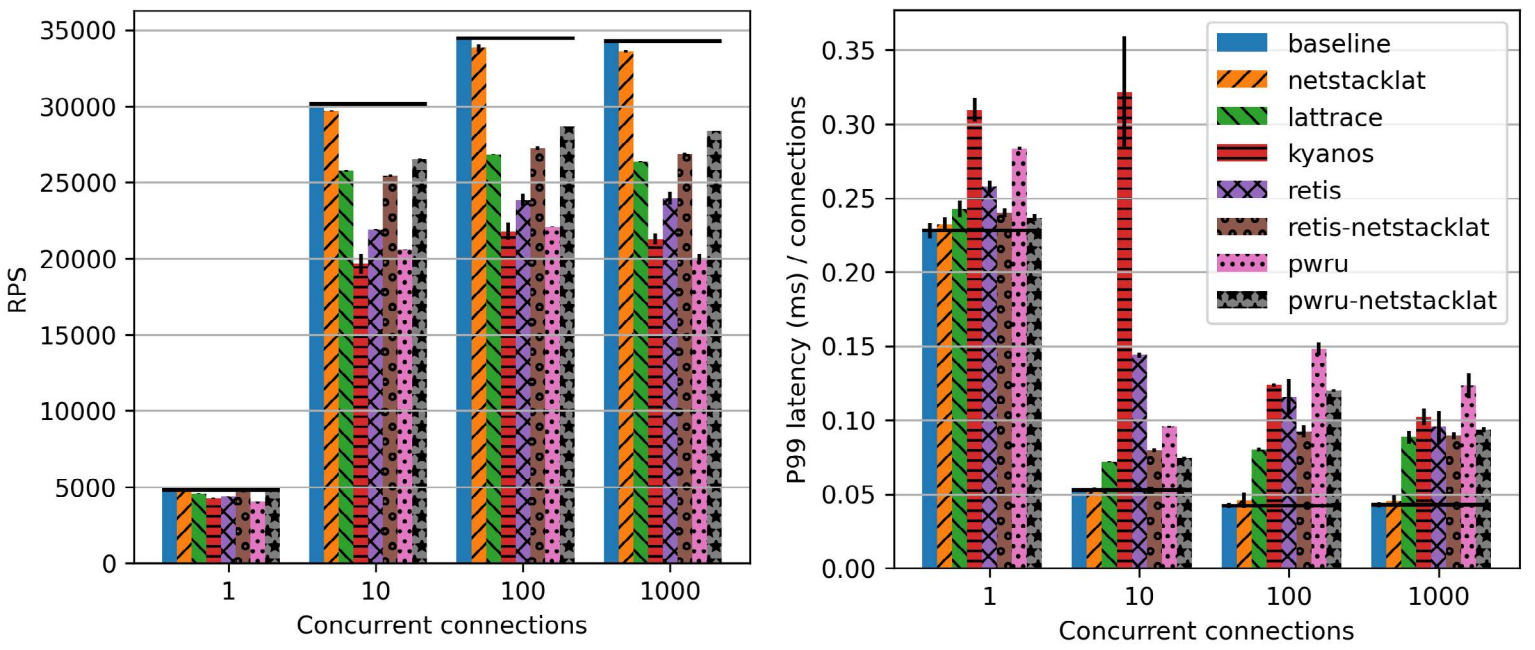
`netstacklat` currently monitors latency at the following four layers of the network stack for TCP and UDP traffic (also shown in Figure 1):

- **ip-start:** The point at which the SKB has reached the start of IP layer processing. Latency at this point comes from early parts of the network stack processing, such as VLAN and traffic control (tc) handling.
- **udp-start and tcp-start:** The SKB has reached the start of UDP or TCP processing. Latency at this point includes IP-layer processing, such as routing and the Netfilter firewall subsystem, in addition to the latency from **`ip-start`**.
- **socket-enqueued:** The SKB has now been enqueued to the application UDP or TCP socket. This is the end of network stack processing, and the SKB payload is now ready to be read by the application. Latency here includes the UDP or TCP layer processing, as well as the latency from **`udp-start`** and **`tcp-start`**.
- **socket-read:** In addition to the network stack latency from the socket-enqueued probes, the latency here also includes any additional delays before the application reads the data from the socket. This may include delays from the application being scheduled to run on a CPU, or the application performing or waiting on other tasks before attempting to read from the socket.

By capturing network stack latency at these four layers, **`netstacklat`** can break down where in the network stack significant latency increases occur. As each probe point reports the cumulative latency up to that point, this breakdown should be performed top-down. If, for example, **`tcp-socket-read`** latency is high and **`tcp-socket-enqueued`** is low, it indicates an issue at the application layer. Meanwhile, if **`tcp-socket-enqueued`** and **`tcp-start`** are also high while **`ip-start`** is low, it indicates that the issue is within the IP layer.

TESTBED EXPERIMENTS

We evaluate **`netstacklat`** in a testbed where we use it to monitor the latency in an nginx server as we run a HTTP load generator against it. **Figure 2** shows how the network stack latency at the four **`netstacklat`** layers grows as we increase the number of concurrent flows with back-to-back requests. We can see that the latency across all layers increases with the number of concurrent connections up to 50 connections. While the large (logarithmic) scale for the y-axis might make the increase for the earlier **`ip-start`**, **`tcp-start`**, and **`tcp-socket-enqueued`** look small, the average latency at 50 or more connections is at least twice as high



RPS

99th percentile response time

Figure 3. *How overhead from monitoring tools impact end-to-end performance*

as for the single-connection scenario. Past 50 concurrent connections, the time it takes the kernel to deliver the SKB to the application socket no longer increases. However, the **`tcp-socket-read`** latency, measuring the time until the nginx application actually reads the request from the socket, continues to grow as increasing numbers of connections create a large request backlog at the application layer. The average time until nginx can start processing the request thus increases from less than 10 microseconds with a single connection to over 100 milliseconds at 4000 concurrent connections.

It is also worth noting that there is a long latency tail at all of the layers, even the early **`ip-start`**. In Figure 2, the 99.99th percentile latency (dotted lines) is typically around an order of magnitude higher than the median latency, even in the lightly loaded single-connection scenario. Across a larger set of workloads we have tested, we have observed maximum latencies being more than three orders of magnitude higher than the minimum latencies at each layer. In extreme cases, this may translate to it taking over 30ms for the kernel to deliver an SKB to the socket, comparable to the end-to-end network latency for a (LEO) satellite link.

We also look at how the overhead from **`netstacklat`** compares to several other tools that can monitor the host network stack latency, namely **`latrace`**, **`Kyanos`**, **`pwru`**, and **`retis`**. The flexible **`pwru`** and **`retis`** tools can be configured to monitor any kernel function receiving an SKB. Here, we test them both with their default (all SKB-related functions for **`pwru`**, and 25 kernel functions in the generic profile for **`retis`**) as well as using the same (smaller) set of kernel functions probed by **`netstacklat`** (**`pwru-netstacklat`** and **`retis-netstacklat`** in the results). This results in six different monitoring setups.

Figure 3 shows the end-to-end performance that our testbed

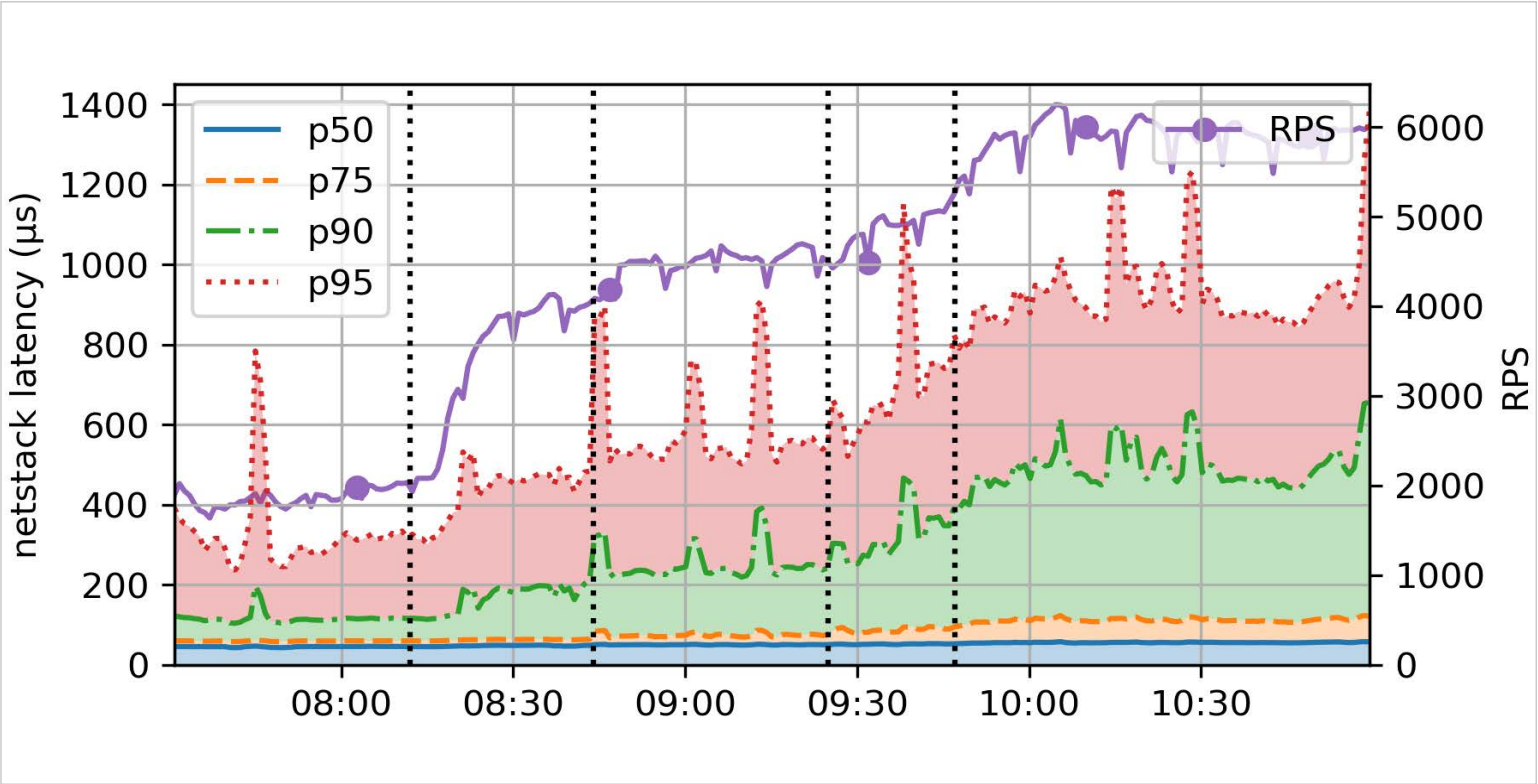


Figure 4. *TCP-socket-read latency under increasing server load*

delivers under each of the six monitoring setups, as well as the baseline performance without any monitoring. Under the heavily loaded scenarios with 100 or 1,000 concurrent connections, **`netstacklat`** reduces the requests per second (RPS) throughput with less than 2% and does not inflate the 99th percentile response time by more than 6% compared to the baseline. In contrast, the other tools reduce RPS by between 17–42% and more than double the 99th percentile response time. Of the evaluated tools, **`netstacklat`** is the only one with sufficiently low overhead to not greatly impact the performance in heavily loaded scenarios, and therefore the only suitable alternative for continuous monitoring.

CLOUDFLARE DEPLOYMENT

The **`netstacklat`** tool has been developed as an open source utility throughout this project. Like many other open source projects, this has led to collaboration across organizational boundaries, highlighting how the open source collaboration model benefits the entire ecosystem. As part of this collaboration, **`netstacklat`** has been successfully deployed on all servers in Cloudflare's global CDN network. As Cloudflare already uses **`ebpf-exporter`** to collect various eBPF-powered metrics, **`netstacklat`** could relatively easily be added to their monitoring pipeline. For this deployment, we focus on the full delay until requests are read by their nginx services, and therefore use only the **`tcp-socket-read`** probe and limit the monitoring to the nginx-related cgroups to minimize overhead.

In **Figure 4**, we show a semi-controlled experiment, where the load balancer was configured to gradually increase the traffic to a server (in steps to 4,000, 4,500, 5,000, and 6,000 RPS at the times indicated by the dashed vertical lines). Here, we can see how the tail latency for the time before the

nginx-ssl instance starts processing requests (**`tcp-socket-read`**) increases roughly proportionally to the request rate. In this instance, the network stack and nginx are not the primary bottlenecks, and a 95th percentile latency below 1ms is unlikely to have a large impact on the end-user perceived delay.

Bottlenecks can shift, however, and **`netstacklat`** has also helped identify instances where the host network latency on some servers increased drastically, with median latencies reaching into the multi-millisecond territory. In that way, **`netstacklat`** helps Cloudflare keep track of where all the requests they serve spend their time. We found the overhead cost for this additional monitoring to be only around 0.04% CPU utilization during peak hours at Cloudflare's busiest location.

FUTURE PLANS

As the networks get faster and resources are moved to the edge, the end host network stack will play an increasingly important role in ensuring low end-to-end network latency. With **`netstacklat`**, we make it possible to continuously monitor how long packets spend waiting on the Linux network stack and applications to process them.

There is, however, still room for improvement. **`netstacklat`** currently only measures the time from the start of the Linux network stack, but studies have shown that significant latency can arise already in the NIC-CPU interconnect. By using the hardware timestamps that many modern NICs can provide, we could extend **`netstacklat`** to measure the total time since the packets first arrive at the NIC. It would also be useful to complement **`netstacklat`**'s high-level latency distributions with detailed traces to simplify root cause identification, and make the selection of points to probe more flexible, similar to **`pwru`** and **`retis`**.

As **`netstacklat`** is open source, it can also benefit from the wisdom and experience of the open source community. If you have an idea for how to improve **`netstacklat`**, please let us know! Or better yet, implement it and open a pull request.

Note: RHRQ includes articles that discuss technologies under active development in upstream open source communities, at universities, and at Red Hat. We want to note that unless otherwise stated, the technologies and how-tos shared here aren't part of supported products, nor promised to be in the future.

About the Authors

Simon Sundberg recently completed his doctorate in data communication and distributed systems at Karlstad University, Sweden.

Anna Brunström is a professor and research manager for the Distributed Systems and Communications Research Group at Karlstad University.

Simone Ferlin-Reiter is a Performance Engineer at Red Hat and adjunct senior lecturer at Karlstad University, Sweden.

Jesper Dangaard Brouer is a kernel developer currently optimizing the network stack and memory allocator in the Linux kernel at Cloudflare.

Toke Høiland-Jørgensen is a senior principal kernel engineer at Red Hat.

MOC ALLIANCE

MAKING THE CLOUD LESS, WELL, CLOUDY

The Mass Open Cloud Alliance (MOC Alliance) is a collaboration of industry, the open-source community, and research IT staff and system researchers from academic institutions across the Northeast that is creating a production cloud for researchers. Of course, a collaboration is only as good as its collaborators.

Follow the MOC Alliance as they create the world's first open cloud.

@mass-open-cloud

www.massopen.cloud

contact@massopen.cloud

RESEARCH QUARTERLY

VOLUME 8:1

Red Hat

Column

Building shoulders for giants to stand on

A long journey, briefly: lessons from a career of building the tools that build new worlds

by Robby Stahl

About the Author
Robby Stahl
is a manager in the Red Hat Research team. His primary focus is platform enablement, driven by a deep understanding of user needs.

Open source and academia have a common axiom: sharing is good. We hope that what we build and discover will be used constructively. As I look back at a couple decades of my career, I see a recurring theme: operating, improving, and building tools for others. I joined Red Hat specifically for the opportunity to hone the frontier of AI tooling into sharper scientific instruments. I truly believe that knowledge should be shared freely. This is how I give back to the society and technology that bring me fruitful joy.

My journey began at a manufacturing company, writing data translation shims between truly ancient systems and slightly less ancient systems. The system administrators declared me competent, and soon I was writing reports against databases that kept the company moving. I realized that I had a solid grasp on the tip of a mountain, and I wanted to understand how it all fits together. I enrolled at a university, and my work became impactful. I maintained and improved a platform that connected students with volunteer opportunities in the local community. This was the first time that I felt like I was using my powers for good.

After a handful of years it was time to move on. I joined the corporate world to see what the pace of business had to offer. I helped keep that world moving by nudging application delivery controllers (ADCs) and web application firewalls (WAFs) toward easier consumption, and hopefully wider adoption. I prototyped interesting things, mainly network functions and software proxies. I ran a lab where people assembled novel and compelling multi-product solutions. A couple years ago I joined a company that wanted to democratize cloud computing. The recurring theme is that I am happy when I build the fundamental pieces that enable larger action, often in ways that I did not intend. I enjoy being catalytic.

Then I heard about a role with Red Hat, and I was hooked by the opportunity to share my talent and effort with academia. I have the privilege of operating, improving, and building tools for researchers and the open source community. It's elative.

PLATFORMS AND PERSPECTIVES

My main goal when managing any platform is that it should be like electricity. A user does not need to know how it works, but they are welcome to ask. A user should never worry if the lights will come on, only if they remembered to turn them off. Even that concern should be minimized where possible. I want to deliver reliability, consistency, availability, and utility.

What do people do with platforms? It turns out that "use someone else's computer" captures most of it: delivering messages asynchronously, offering uninterruptible services, producing

artifacts, analyzing vast amounts of data. In other words, things that cannot be reasonably accomplished on a personal machine. Most importantly, people collaborate. They share. They build. In my experience, this is surprisingly consistent between private industry and public institutions. The difference is that companies do not default to sharing their findings freely, while public institutions strongly prefer to share with the world.

> Computers work so people can learn, explore, and play.

There is also an implicit goal for every platform: reduce human toil. This is my favorite part of the problem space. Computers work so people can learn, explore, and play. Reducing the time and effort to start a project from "I need to buy my own hardware, then install and configure software, and then..." to "I need to open a ticket" is empowering. More importantly, a well-positioned platform lowers the barriers to entry and catalyzes research. I need merely to glance at my heap of half-completed projects to understand the importance of easy access to tools and resources.

The possibilities are fascinating: I can delegate tasks to an agent using reasonably plain language. With the advent of agentic and multimodal frameworks, I just might start addressing a computer by name, a la Star Trek. I must consider a fundamental change in how I interact with systems. The difference is that instead of asking a device in my house to turn down the volume, I can ask an agent to optimize a CI/CD pipeline for minimum energy consumption at the expense of wall time. I can ask an agent to be my "rubber duck" as I talk through a self-inflicted technical issue.

Neat.

WE STAND ON THE SHOULDERS OF GIANTS

Our world is a cumulative effort. Writing, art, and now digital information allow civilization to create, discover, and improve generation-by-generation, second-by-second. The metaphor, standing on the shoulders of giants, is not new. However, the velocity with which we are building layers of giants is unprecedented.

We are approaching, or possibly passing through, an inflection point where giants are not necessarily human. Is it appropriate to cite an LLM as a co-author for a paper? Is the training set a required citation for AI assistants performing data analysis?

Wrapping a framework around models can reduce the time required to test a hypothesis. Karpathy's autoresearch method uses models to experiment with tuning models. The options for using agents to collate and synthesize are legion. Models are fantastic tools, but they are neither responsible nor accountable.

Raw creativity and inspiration are still gifts of the human experience. Collaboration is a fundamental human drive. For now, the state of the world is that giants can enlist demi-giants to accelerate their work.

OPERATING SHOULDERS-AS-A-SERVICE

The current state of training and inference is that researchers, models, and agents require more computing power than most people can afford to own outright. The computing world is defining and exploring architectures and paradigms that can change consumption patterns monthly.

From a platform perspective, I must be aware of, yet not beholden to, the usage patterns. I do need to guarantee that workloads cannot impact their neighbors. I must ensure that hardware is running nominally. I must verify that the software components are secure and updated.

The interesting part, for me, is the research itself. Technical explorations tackle problems like combining multiple agents for multi-modal solutions, or exploring the integration of model-as-a-service with CI/CD practices. We explore the boundaries of using open models in place of closed models. And yet, these are all tools that scientists in other domains may use to advance their research. That is the aspect that brings me the most joy. Perhaps a revolutionary medical imaging technique will be developed using something I built.

The Mass Open Cloud, the open source community, the interconnected world—it is all so much more than any person could accomplish in a lifetime. A horde of nerds is a fantastic force of invention. I am truly happy to be a part of it.

Every researcher stands on the shoulders of giants and lends their shoulders to the next tier. I get to build an open laboratory for giants.

UMass Lowell is proud to collaborate with Red Hat, a Select Preferred Partner, and celebrate more than a decade of working together on research, philanthropy and building the next generation of Red Hat's workforce.

UMASS LOWELL

GET A LAPTOP THAT IS AI READY

AMD RYZEN™ AI TECHNOLOGY IS NOW BUILT IN

AMD
RYZEN AI

AMD RYZEN 7000 SERIES
AMD RADEON GRAPHICS

*Available on selected systems