RH RQ

Bringing great research ideas into open source communities

Luke Hinds

"It's the wild frontier": security, agentic AI, and open source

+

An energy-aware, AI-ready data space

Efficient, transparent GPU scheduling

Yuga for Rust analysis

Closing the AI gap

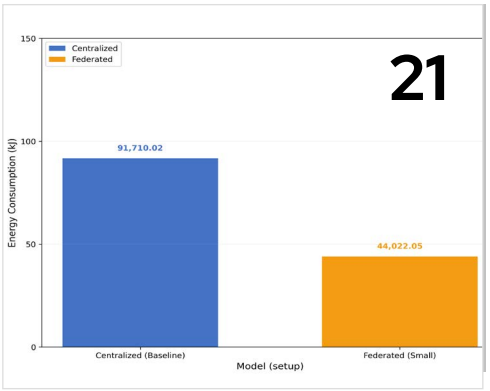
Red Hat Research Quarterly
Volume 7:2 | Fall 2025 | ISSN 2691-5278

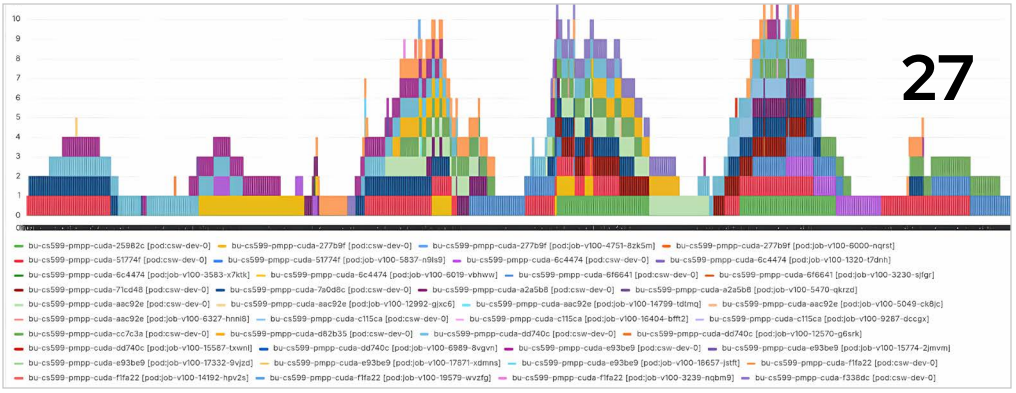
RESEARCH QUARTERLY

VOLUME 7:2

Red Hat

Table of Contents





ABOUT RED HAT Red Hat is the world's leading provider of open source software solutions, using a community-powered approach to provide reliable and high-performing cloud, Linux®, middleware, storage, and virtualization technologies. Red Hat also offers award-winning support, training, and consulting services. As a connective hub in a global network of enterprises, partners, and open source communities, Red Hat helps create relevant, innovative technologies that liberate resources for growth and prepare customers for the future of IT.

NORTH AMERICA
1 888 REDHAT1

EUROPE, MIDDLE EAST, AND AFRICA
00800 7334 2835
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
info-latam@redhat.com

facebook.com/redhatinc
@RedHat
linkedin.com/company/red-hat

Red Hat

RESEARCH QUARTERLY

VOLUME 7:2

From the director

Closing the AI gap: why we can't leave students—or Montana—behind

About the Author

Heidi Picher Dempsey is the US Research Director for Red Hat. She seeks and cultivates research and open source projects with academic and commercial partners in operating systems, hybrid clouds, performance optimization, networking, security, and distributed system operations.

by Heidi Dempsey

In the film *The Hunt for Red October,* a Soviet submarine captain intends to defect to the United States with his state-of-the-art nuclear submarine, and he discusses plans with his senior officers while underway. "I will live in Montana," one says, and I will marry a round American woman and raise rabbits, and she will cook them for me. And I will have a pickup truck... maybe even a 'recreational vehicle.' And drive from state to state. Do they let you do that?"

Montana gladly supports these freedoms, and more. For some reason, however, it is the only state in the US not currently participating in the National AI Infrastructure Research Resource (NAIRR) Pilot program (nairrpilot.org/map). Sponsored by the National Science Foundation (NSF), this program aims to connect people to the resources they need to "advance AI research and the research that employs AI." No slight to the great state of Montana—we don't know why Montana abstained or was overlooked in this instance. But the gap points to a growing issue with AI research and education: even though AI will impact the lives of every person in the US, we have no robust program for training all our students to understand and be able to affect AI.

We also have very few efforts underway to make clear to average citizens what AI does, how it works, and why it makes decisions that affect their lives. Even when an organization like the NSF undertakes an expensive multi-year effort (540 projects!) to address some of the blockers that researchers and students experience when trying to better understand and control AI technology, it is still very easy for large groups of people to be overlooked. How much easier is it to overlook the educational, social, regulatory, and political work necessary to truly "democratize" AI? The United States AI Action plan, published in July 2025, only begins to scratch the surface of these areas. Just as the drive for an open operating system powered the early days of Linux research, creating an open and transparent way to use (or not use) AI must drive the work that industry, government, and ordinary people do as we adopt AI in more and more places in society.

Red Hat and IBM are working together to support NAIRR pilot projects (more on this in the next issue) in the Mass Open Cloud. This work is global, of course, even though I happen to be only writing about US efforts in this article. A quick look at research.redhat.com as well as the IBM and Red Hat corporate blogs highlights many AI infrastructure projects in EMEA, South America, India, Australia, and the Far East. Red Hatters in the Czech Republic are actively studying the AI Act requirements for EU R&D as well as new

infrastructure and adapting our technical goals to comply with new regulations.

This work also extends to projects like TrustyAI, which provides tools for responsible AI workflows, and the AI Bill of Materials, or AI BOM. This Bill of Materials for an AI system could provide a common structure and tools for recording details about which model versions, datasets, and tools are used to create AI that drives applications like chatbots, no matter who originally produces them. Think of it like a standardized "Nutrition Facts" label for AI instead of foods.

Basic information like this is important for us to be able to understand more about how a particular AI was developed and trained to answer our questions. The interview with Sigstore founder Luke Hinds in this issue of RHRQ also highlights new efforts to equip AIs with explainable and provable security, and to design them to always consider energy conservation and climate effects as part of AI application design.

Finally, Red Hat Research has been pursuing several efforts to optimize the performance of AI engines by tuning the many "knobs" in the full stack for compute, networking and storage sub-systems that must work together efficiently in an AI datacenter. Even when performance optimization is the goal, there are still many tradeoffs in options to consider, from the BIOS all the way up through the stack to controlling the placement of nodes in clusters, and the interactions of AI agents through MCP. You'll be hearing more about these in future issues as well.

Of course I am only mentioning a tiny fraction of the important work we all must pursue here. Similarly, only a tiny fraction of the people who need to be brought in to do this work are currently able to access it—only 0.01% of the projects currently underway in NAIRR have education and training as their primary goals. Only 2.5% of colleges and universities offer a BS degree in AI. According to OpenMined, AI itself is trained and evaluated on less than 0.01% of the world's data. We are overlooking much more than just Montana.

News

The post-quantum cryptography transition: researching a quantum-safe future

Collaborative research among industry, academia, open source communities, and government is proactively developing quantum-resistant solutions.

by Martin Ukrop and Arthur Savage

About the Author

Martin Ukrop is a Principal Research Software Engineer with Red Hat Research, focusing on security research and facilitating industry-academia cooperation in EMEA.

About the Author

Arthur Savage is a software engineer on OCTO Emerging Tech's Security Team. He has a Master's degree in Electrical and Computer Engineering and specializes in cryptography, data analytics, and image forensics.

The rise of large-scale quantum computers presents a direct threat to the cryptographic primitives that secure our most critical digital infrastructure. As these machines advance, they will be capable of breaking algorithms like RSA and ECC, which form the bedrock of modern public-key cryptography. Global regulatory momentum has already driven a significant acceleration in security research. The US National Institute of Standards and Technology (NIST) finalized its first set of post-quantum cryptography (PQC) standards in August 2024, and the development and drafting of RFCs for the Internet Engineering Task Force (IETF) are now underway, with federal agencies receiving mandates to begin the migration process immediately. These new standards will soon be required for use in commercial and government systems, leading up to meeting the target of a broader government-wide transition to PQC no later than 2035—although security experts suggest that Q-Day may come closer to 2030. In either case, there is a great deal of work to do in the next few years.

Engineers from Red Hat's Office of the Chief Technology Officer (OCTO) are collaborating on two recently launched projects with the aim of preparing for this transition: Quantum-Resistant Cryptography in Practice in EMEA and the introduction of frameworks to enable future PQC software signing as part of the upstream open source project Sigstore.

QUANTUM-RESISTANT CRYPTOGRAPHY IN PRACTICE

The Quantum-Resistant Cryptography in Practice (QARC) project is an initiative under the European Union's Horizon Europe program*, coordinated by the Brno University of Technology, set to run for three years starting in early 2026. The project aims to create a robust and practical framework for the transition to PQC. It brings together a diverse consortium of partners from 11 European countries, representing three key sectors: industry, academia, and governmental organizations. The objective is to move beyond theoretical research and develop real-world, secure implementations of

quantum-safe cryptographic algorithms for complex, sensitive applications like e-voting, cloud services, and Linux authentication. Cryptographic agility is a key element of the project, enabling systems to adapt quickly to new standards and threats.

Another impactful component is hands-on application through practical pilots. Pilots will test PQC implementations in real-world scenarios including e-government services in Estonia, cloud services by the Latvian company Tet, and Linux authentication for enterprise. QARC will also address the broader, non-technical challenges of PQC transition by establishing an international network of National Cybersecurity Authorities to coordinate national strategies and provide harmonized recommendations. The goal is to generate open source software, hardware designs, and best practices to help a wide range of organizations navigate the shift to a post-quantum world.

Red Hat is a major contributor in the QARC consortium, leveraging engineers' experience in open source software and standards. Contributions center on the practical implementation and standardization of PQC in widely used open source ecosystems, with a specific focus on core open source cryptographic libraries like OpenSSL, GnuTLS, and NSS. Their work involves extending the `tlsfuzzer` test suite to detect vulnerabilities in these new algorithms, ensuring the implementations are robust and secure. Red Hat engineers lead the PQC Linux Authentication pilot, which aims to make the Kerberos authentication protocol quantum-safe, adapting existing protocols to work with new PQC standards and prototyping changes to MIT Kerberos based on these updates. We will also integrate post-quantum Public Key Infrastructure into FreeIPA, its identity and authentication solution for Linux, allowing it to generate and manage PQC certificates. The findings from this pilot will provide crucial insights into how to deploy quantum-safe cryptography in existing enterprise workflows.

PQC SOFTWARE SIGNING

In early March, the Red Hat Emerging Technology Security Team embarked on a project to introduce PQC to the secure software supply chain in collaboration with the Red Hat Trusted Artifact Signer (RHTAS) team. Currently, the team works with Sigstore, an open source project providing the tools to sign software and artifacts then publish proof of lineage to a transparency log for simple, non-repudiable verification. Though Sigstore is widely adopted by industry and government entities—those who need PQC most—it has lagged behind on PQC adoption, in part due to lack of signature support in core upstream packages like Go standard cryptographic library, as well as delays in finalizing worldwide cryptographic standards.

We've had to get creative and stay flexible. Much of our time was spent just negotiating plans, and plans made one week will be different the next, depending on the wider PQC ecosystem: changing government regulations, industry needs, emerging academic research and weaknesses, and volatile APIs. It's a fascinating, exciting space to work, and we're making real progress. Portions of our software design have already been merged upstream with much more to come.

Towards the end of the year, the team hopes to have a proof-of-concept for PQC software signing with Sigstore, functionality that will then be expanded and incorporated into RHTAS. Contributing these features directly to the upstream community accelerates industry-wide innovation and ensures Red Hat continues its long legacy of open source contributions and collaboration, while keeping our products at the bleeding edge of mitigating one of the greatest existential cybersecurity threats of the modern day. In the future, the Emerging Tech team will be investigating how to maintain the hardware root of trust in a post-quantum world—a matter critical for confidential computing and zero-trust architecture.

STRENGTH THROUGH OPENNESS

The imperative to transition to post-quantum cryptography is no longer a distant concern but an immediate necessity, driven by rapid advancements in quantum computing and evolving regulatory mandates. Collaborative initiatives like the Quantum-Resistant Cryptography in Practice (QARC) project and the integration of PQC into Sigstore for software signing demonstrate the importance of solving PQC problems in the open. Engaging a diverse set of stakeholders will fuel more rapid innovation, create greater transparency, and ensure wide access to security solutions.

***QARC** is to be funded by the European Union under Grant Agreement No. 101225691 from 2026.*

Funded by the European Union

Publication highlights

Research collaborations between Red Hatters and key Red Hat industry and university partners often produce peer-reviewed publications that bring open source contributions along with them. These research artifacts illustrate the value that open industry-academia collaborations hold not just for participants, but for technological advancement across the field of computer engineering. This is a sampling of recent papers and conference presentations; to see more visit the publications page of the Red Hat Research website (research.redhat.com/publications).

"Analysis of smart imaging runtime," Thomas Athey (Massachusetts Institute of Technology), Shashata Sawmya (MIT), Yaron Meirovitch (Harvard University), Richard Schalek (Harvard), Pavel Potocek (Thermo Fisher Scientific, Saarland University), Ishaan Chandok (TFS, Harvard), Maurice Peemen (TFS), Jeff Lichtman (Harvard), Aravinthan Samuel (Harvard), Nir Shavit (MIT, Red Hat) In (2025) *Applied Microscopy*, 55(1), art. no. 10.

"AnnotationGym: a generic framework for automatic source code annotation," Hafsah Shahzad (BU), Ahmed Sanaullah (Red Hat), Sanjay Arora (Red Hat), Ulrich Drepper (Red Hat), Martin Herbordt (BU). In (2025) *IEEE Access* 13, pp. 155321-339.

"Automating the detection of code vulnerabilities by analyzing GitHub issues," Daniele Cipollone (Delft University of Technology), Changjie Wang (KTH Royal Institute of Technology, Sweden) Mariano Scazzariello (RISE AB), Simone Ferlin (Red Hat), Maliheh Izadi (DUT), Dejan Kostic (KTH, RISE AB), Marco Chiesa (KTH). In (2025) *Proceedings of the 2025 IEEE ACM International Workshop on Large Language Models for Code Llm4code*, pp. 41-48.

"Best-effort power model serving for energy quantification of cloud instances," Sunyanan Choochotkaew (IBM Research), Tatsuhiro Chiba (IBM Research), Marcelo Amaral (IBM Research), Rina Nakazawa (IBM Research), Scott Trent (IBM Research), Eun Kyung Lee (IBM Research), Umamaheswari Devi (IBM Research), Tamar Eilam (IBM Research), and Huamin Chen (Red Hat). In (2024) *32nd International Conference on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

"Bridging clusters: a comparative look at multicluster networking performance in Kubernetes," Sai Sindhur Malleni (Red Hat), Raúl Sevilla (Red Hat), José Castillo Lema (Red Hat), André Bauer (Illinois Institute of Technology). In (2025) *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering*, pp. 113-23.

"Efficient manipulation of control flow models in evolving software," Tomáš Fiedor (Brno University of Technology, Red Hat), Jiří Pavela (BUT, Red Hat), Adam Rogalewicz (BUT), Tomáš Vojnar (BUT, Masaryk University). In (2025) *Lecture Notes in Computer Science* 15172 LNCS, pp. 412-28.

"Fixing invalid CVE-CWE mappings in threat databases," Şevval Şimşek (BU), Howell Xia (BU), Jonah Gluck (BU), David Sastre Medina (Red Hat), David Starobinski (BU). In (2025) *IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 950-60.

"Generic multicast," José Augusto Bolina (Red Hat), Pierre Sutra (Télécom SudParis, INRIA, France), Douglas Antunes Rocha (Federal University of Uberlandia, Brazil), Lasaro Camargos (FUU). In (2024) *ACM International Conference Proceeding Series*, pp. 81-90.

"A graph-based algorithm for optimizing GCC compiler flag settings," Reza Sajjadinasab (Boston University), Sanjay Arora (Red Hat), Ulrich Drepper (Red Hat), Ahmed Sanaullah (Red Hat), Martin Herbordt (BU). In (2024) *IEEE High Performance Extreme Computing Conference (HPEC)*.

"Green.Dat.AI: a data spaces architecture for enhancing green AI services," Ioannis Chrysakis (Ghent University, Belgium) , Evangelos Agorogiannis (Netcompany-Intrasoft, Luxembourg), Nikoleta Tsampanaki (Netcompany-Intrasoft), Michalis Vourtzoumis (Netcompany-Intrasoft), Eva Chondrodima (University of Piraeus, Greece), Yannis Theodoridis (Piraeus), Domen Mongus (University of Maribor, Slovenia), Ben Capper (Red Hat), Martin Wagner (Eviden), Aris Sotiropoulos (AEGIS IT Research), et al. In (2025) *Design, Automation & Test in Europe Conference (DATE)*.

"A neural network based GCC cost model for faster compiler tuning," Hafsah Shahzad (BU), Ahmed Sanaullah (Red Hat), Sanjay Arora (Red Hat), Ulrich Drepper (Red Hat), Martin Herbordt (BU). In (2024) *IEEE High Performance Extreme Computing Conference (HPEC)*.

"Performance comparison of service mesh frameworks: the MTLS test case," Anat Bremler Barr (Tel Aviv University), Ofek Lavi (TAU), Yaniv Naor (Reichman University), Sanjeev Rampal (Red Hat), Jonathan Tavori (TAU). In (2025) *Proceedings of IEEE IFIP Network Operations and Management Symposium (NOMS)*.

"PraxiPaaS: a decomposable machine learning system for efficient container package discovery," Zongshun Zhang (BU), Rohan Kumar (BU), Jason Li (BU), Lisa Korver (BU), Anthony Byrne (Red Hat), Gianluca Stringhini (BU), Ibrahim Matta (BU), Ayse Coskun (BU). In (2024) *Proceedings of the 2024 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 178-88.

"Scalable and distributed processing of 3D astronomical data cubes for galaxy evolution studies using the AC3 framework," Mario Chamorro-Cazorla (Universidad Complutense de Madrid, Spain), Christina Catalán-Torrecilla (UCM), Ray Carroll (Red Hat), Ben Capper (Red Hat), Ryan Jenkins (Red Hat). In (2025) *11th IEEE Conference on Network Functions Virtualization and Software-Defined Networking (NFV-SDN'25)*, forthcoming.

"Timerlat: real-time Linux scheduling latency measurements, tracing, and analysis," Daniel Bristot de Oliveira (Red Hat), Daniel Casini (Scuola Superiore Sant'Anna, Italy), Juri Lelli (Red Hat), Tommaso Cucinotta (SSSA). In (2025) *IEEE Transactions on Computers* 74:8, pp. 2608-20.

"Understanding similarities and differences between software composition analysis tools," Pranet Sharma (BU), Zhenpeng Shi (BU), Şevval Şimşek (BU), David Starobinski (BU), David Sastre Medina (Red Hat) In (2025) *IEEE Security and Privacy* 23:1, pp. 53-63.

"VeBPF many-core architecture for network functions in FPGA-based SmartNICs and IoT," Zai Tahir (BU), Ahmed Sanaullah (Red Hat), Sahal Bandara (BU),Ulrich Drepper (Red Hat), Martin Herbordt (BU). In (2024) *IEEE High Performance Extreme Computing Conference (HPEC)*.

"Wasserstein distances, neuronal entanglement, and sparsity," Shashata Sawmya (MIT), Linghao Kong (MIT), Ilia Markov (IST Austria), Dan Alistarh (IST Austria, Neural Magic/Red Hat), Nir Shavit (MIT, Neural Magic/Red Hat) In (2025) *13th International Conference on Learning Representations (ICLR)*, pp. 26244-274.

Open dissemination of research results increases transparency, promotes replicability, and acclerates innovation. Share with us by reaching out to academic@redhat.com.

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

"It's the wild frontier"

Security, agentic AI, and open source

*An interview with **Luke Hinds***

*conducted by **Ryan Cook***

Red Hat

Interview

The name Luke Hinds is well known in the open source security community. During his time as Distinguished Engineer and Security Engineering Lead for the Office of the CTO Red Hat, he acted as a security advisor to multiple open source organizations, worked with MIT Lincoln Laboratory to build Keylime, and created Sigstore, a wildly successful open source project for improving software supply chain security that quickly became the standard for signing software components. You don't have to spend long with Luke to realize he has a restless mind and a strong commitment to open source development and communities. Since Sigstore, he's channeled that passion into co-founding the open source startup Stacklok, where he remained as CTO until May 2025, and launching AgentUp, an open source framework to help developers build interoperable AI agents quickly, flexibly, and securely. As he'll explain below, you might think of it as Docker for agentic AI.

RHRQ asked Ryan Cook, the platform and Enterprise AI lead in the Red Hat Emerging Technologies group, to lead a wide-ranging conversation with Luke. Together they discuss the urgent need to develop security in AI, the importance of model provenance and transparency, the essential role of the open source community, and adapting authorization protocols for AI agents. –Shaun Strohmer, Ed.

About the Interviewer

Ryan Cook is the platform and Enterprise AI lead in the Emerging Technologies group at Red Hat.

Ryan Cook: Let's be honest: security is often the last thing developers think about. Now add AI—it is a whole different world in terms of speed, and it seems we're just now catching up in the security space. What are you seeing?

Luke Hinds: Security is barely there at the moment. Obviously some people are more advanced than others, but it's very nascent. There are these large frontier models coming out of Anthropic, OpenAI, and Gemini. Some models claim they're open, but they're not; the datasets are not open. These are black boxes: there's no way of measuring these things. There's no way of testing them because they're never determined. Input comes in and a variant output will always come out, so there's no way of understanding the true nature of these things. That gets dangerous from a security angle. There are prompt injection attacks, but people can also weaponize these models such that the weights and biases in the neural network are heavily influenced to act a certain way. It could behave in a different way from one group of people to another, or one language to another. That's one scary part of security in AI. For people who've been working in security for years, everything's been determined. If there's a bug, you have an IDE with a breakpoint, and you can inspect the stack trace and the variables, and if you run it again, everything will be the same every time. With AI, it's never like that. It has that propensity to act differently every single time.

This is also where people are having problems with agents, which I've been looking at for quite

In his off hours, Luke is a long-distance runner. "When I run, my brain goes into the default mode network," Luke says. "I have a lot of ideas when I'm running—I'll pull over and rant into my dictaphone."

a while. They never operate in the same way every time. There'll always be some slight variance— sometimes it might be very wide variance and a hallucination. That's the big thing on the model security side, which is where model provenance comes in. That's where Sigstore's got a second wind recently, and Red Hat has been working a lot on it as well. You want to be able to look at a model and know this dataset is what built this model— the model's DNA, or genome structure. Then you can see there's obviously a bit of advertising thrown in, or there's a little bit of something leaning to the right or the left, or there's plain misinformation. Model provenance is really important— and this is getting into Red Hat territory. Open models are massively important, not just for transparency and the community working together, but for safety. Again, due to this probabilistic nature, you can't rely on them to perform or act a certain way, especially when they're calling tools and making their own decisions.

Ryan Cook: There's a lot of room for it to go astray. Everybody has been talking about the importance of humans in the loop, but it also goes back to open models and having the insight into what actually generated it to ensure you don't have a biased model. I know there's a couple of bigger communities that are fully, absolutely transparent with their models, and I hope we get to a point where those are much more public and used much more. The space is well on its way to getting better, but there's still a lot of room to grow.

AGENTUP

Ryan Cook: Speaking of room to grow: Agent Up. What sort of opportunities have you seen in that space? What were your goals when you started coming up with that project?

Luke Hinds: I was building AI agents out of curiosity: writing from the ground up, thinking about what an agent would look like. I spoke to other people about their ideas of what a good agent looks like, and the consensus was that there was little guidance around how to do things the right way, even for fundamentals that are not necessarily AI-centric: security, rate-limiting, persistence, distributed scale, performance. They've been using some of the existing frameworks, and I heard this repeated pattern of frustration with the lack of a clean set of interfaces to work with. It's a nascent space—the wild frontier.

So I had the idea to build something where people could quickly bootstrap an agent with good old essentials in there. I started to play around, and I realized that an interesting direction would be to make something that's portable. I've always looked at Docker as a brilliant example of something reproducible and portable, and you can bring it up to a good standard quickly. You have this contract you can pass around—a Docker file—and it can go into a GitHub repo. Then people can pull it and run it and have exactly the same environment as everybody else. I thought that would be a nice thing to have for agents. My attempt with AgentUp was to build that and have that config-driven type of experience.

With AgentUp, there's a framework, but you don't build on top of an SDK framework like you would normally—you use entry points. You can write as much customized code as you like, but it's all managed as dependencies, which was something else I came across by mistake because I couldn't get something to work. It's at an interesting place now, figuring out where to take it next, because it does have that portable, reproducible, pinnable structure, which is really resonating with folks, because then you can have something that runs the same on a developer's laptop as it does in production.

So right now, Agent Up is at a similar stage to where Sigstore was for a while. I'd be talking to people about Sigstore and you could tell they were looking at the other monitor, pretending to follow along while they're thinking, "I don't know what this guy's talking about." Although I think AgentUp's a bit further along, because I'm hearing from folks like you who are clicking with it, saying "I see where this is going."

Ryan Cook: I definitely feel like you won my heart from a sysadmin point of view. You've been in the game long enough and you remember the days when you'd say, "Hey, it worked on my laptop, but it doesn't work on the server or it doesn't work on my friend's laptop."

Luke Hinds: Absolutely. It was a mess trying to have all these different versions on your machine. That was a great thing with containers: they solved a real pain. That's another thing I've learned about successful open source projects. I will ask myself, "Luke, is this a painkiller or a supplement?" Supplements are nice. They're a good idea. You take a multivitamin and it's very easy. But if you leave home and you forget to take your vitamin D, you don't turn the car around and race back home. You think, "I can just take it tomorrow."

Whereas if you've got a migraine or some real pain, you need a painkiller. Everything else is on hold until you get into Walmart or you turn the car around. Even though you've gone 15 minutes down the road, you go back and get it because you got a pain to solve. I always try to find projects that lean towards solving a pain, but they're still easy. You throw them in your mouth and you drink. If you have a project that's a supplement, and it takes about a week of trying to read horrible, obscure documentation to get it to work, your project's never going to get anywhere. Your startup's going to die.

> Open models are massively important, not just for transparency and the community working together, but for safety.

"ANYTHING IS POSSIBLE"

Ryan Cook: How did you decide you want to work with computers? For example, I took apart the keyboard of my computer and put it back together, and a week later, my mom said, "You're going to start early college classes on computers." That was it.

Luke Hinds: I was pretty similar. There was a computer game at the time called Elite, with all kinds of vector graphics, and you'd go around these universes and you'd mine rocks and alloy. But it was only available on the BBC Micro—the BBC had their own computers that were in all the schools. I really wanted to play, but my single-parent mom couldn't afford a computer. To get one I had to join the school computer club, but first I had to prove my worth. I wrote my first program on paper! I tried to write an adventure game, and it got very out of hand quickly—my wrist hurt from writing so much. I showed it to the math teacher, and he was impressed I'd even tried, so even though it was a mess he let me in. I owe that math teacher quite a lot really, because that meant I got my hands on a computer for the first time. From there it just got out of control, really.

Ryan Cook: Where did you want to go with it?

Luke Hinds: I didn't really know. I started off in hardware, initially doing repairs on computer boards—a lot of poking around with an oscilloscope and soldering chips on and off. Then a friend of mine joined a company as a software engineer and then I went there as a support engineer. I was constantly like, "I want to do what they're doing, writing the code. That looks more interesting." I think if you have a passion about this you're going to find a way. Coding was the thing that really made me home in on a particular area, because I love the creativity of it. It's just amazing: you can write stuff and get a computer to do something.

Trying to get developers to adopt security is like trying to get a toddler to eat their greens. They may agree that rationally it makes sense, but they just don't want to.

Ryan Cook: I completely agree. When I started actually writing code it was like, "Oh my gosh anything is possible." So now that we're seasoned engineers, what do we tell the next generation of folks, especially about getting into open source or AI?

Luke Hinds: About open source, I would say, first, it's very good for your career, because you get public exposure of what you're doing. If I'm in a position to be hiring engineers, I'll want to find their GitHub and see what they've done. I'm not expecting perfect code, but if I see somebody trying stuff, that's a really good signal. Second, you can mingle with very senior folks within a community. A lot of communities are very accepting of first-time contributors. I love it when somebody new turns up. One of the things I always do in my projects is mentoring, even if that's helping someone do their first PR or figure out how to use Git. They can watch the project developing, see how problems are addressed in open discussions, and then see the code to address them.

I do fear for the younger folks, because they are coming into a world where coding assistants and AI tools let you knock out a project in 20 or 30 minutes. The conditions are not there to force people to learn. About 10 years ago I tried to write an OS in Rust. It sounds impressive but it's really not—it didn't go very far. But I remember spending four days just banging my head against the wall trying to figure things out. It forced me to learn something I wouldn't have learned about otherwise. If necessity is the mother of invention, but people have this instant knowledge available with no need to work at it, where will the inventions come from?

Another thing I've noticed being in the agent space is an absolute tidal wave of vibe-coded projects. You can tell straight away because the READ ME is full of emojis and rockets. You've got these projects where the first commit was six hours ago and there's a READ ME talking about it being enterprise grade and some of them even quoting SLAs, and it's just an LLM just spitting this stuff out. That also makes it a challenge to get your stuff a bit more to get noticed. For somebody younger that's writing something genuinely useful, being able to rise above the slosh out there can be tough.

Ryan Cook: One of my big concerns is whether it's working in a best practice way, security-wise or otherwise. What do you think is the best way to utilize those vibe-coding situations while keeping best practices in mind? And what is the best way for someone younger in their career to utilize those services and still learn something?

Luke Hinds: I would say turn off auto-accept, read what it produces, and ask it to explain what it's doing. It might reveal that it doesn't quite understand. Ask, why did you do that—why did you choose that dependency? Why did you make that design choice? I use Claude and similar tools a lot because they are absolutely brilliant for prototypes, and another thing I've noticed is you'll get some pretty good code and it appears to work really well. But when the system gets above a certain level of complexity, you get underneath and you realize, this is so brittle. It's like a toy train

going around the track, but half the track is missing. The train goes off the track, along the baseboard of the wall, then comes back on the rails. It looks like it's working, but it's not.

Ryan Cook: You bring up an excellent point. Tying that back to open source and GitHub: somebody early in their career could use vibe coding, question everything, and learn their way up, and also have that public GitHub repository and demonstrate they know how to use Git and write code and understand it. You can build out your knowledge set portfolio with these two things together and come into an interview and just absolutely rock it.

Luke Hinds: Absolutely. Otherwise, you can build some impressive projects with AI, but when you sit down with a group of people and they start asking about event-based systems, distributed systems, and what did you use for your queueing system, it's going to become very clear you don't know what you're talking about. And don't feel you're going to be replaced. I don't believe in this "AI is going to replace all software engineers" noise. There are things agents are great at. They love open-ended stuff where they get to choose the goal. But if you want them to deliver the same goal every time, they can't.

Ryan Cook: I like to say that with some agents and some of these LLMs, I've never been more correct in my life than when using them, and I've never been more incorrectly correct.

THE POWER OF COMMUNITY

Ryan Cook: Moving forward in your career, you built one of the biggest supply chain securing projects that exists—you created an entire ecosystem. How did that come about?

Luke Hinds: So, I'd been thinking about software supply chain security for some time, being in security, and that term "software supply chain security" was starting to bubble up. I'd come across transparency logs, which is something they used for cryptographic guarantees around who created a certificate for whom. (CoreOS co-founder) Brandon Phillips was digging around this area as well, and I remember talking to him and thinking, well, I'm just going to try to write something. I'm going to build a prototype: I'm going to get this transparency log and try to start putting signatures of artifacts in there.

I think for Sigstore it was a case of "right project, right time." I've developed lots of right projects-wrong times and I've developed lots of wrong projects-right times, so I was quite lucky in this case. There were a lot of people looking to solve this problem, and they just converged on what I was doing. Something similar happened with Linus (Torvalds, creator of Linux) originally. I'm not comparing myself to Linus at all, but it was similar. He popped up on Usenet saying, "Hey, I got this thing, not really quite sure where to go with it." And then other people were like, "Well, I would like to work on it." It was very much the same with Sigstore. What I maybe brought to it was the vision of where it could go, because it could have just been people working on something without having a long-term trajectory.

I used to call it the Let's Encrypt for software signing: it would be a public-good, vendor-neutral service so everybody could sign things: a 12-year-old kid in his bedroom or the corporation with billions of dollars. I realized if it was going to be successful, first, it had to be a public-good, neutral service and second, it had to be very simple to use. One thing about security: trying to get developers to adopt security is like trying to get a toddler to eat their greens. They may agree that rationally it makes sense, but they just don't want to. They've got all these APIs and AI saying, "We'll make you faster and better," so it's hard to get the security story to land unless they really don't have to do anything.

Ryan Cook: You brought up getting the community behind Sigstore. Even with making this almost a free service, how do you feel the community and making those things available in the open changed the project and made it easily adoptable? There was probably a possibility for us at Red Hat to just put the lock on the door—even though Red Hat doesn't do that—so did it make a difference to start open?

Luke Hinds: Oh, it massively made a difference. If Google had not got involved and others, it likely wouldn't have gone anywhere near as far as it has, and Google is still heavily involved. I remember going to (Red Hat CTO) Chris Wright and saying, "Hey, I want to put this in the Linux Foundation. I have this idea of a public-good service," and he could see the picture. It would not have been a success if we hadn't taken it to the community.

Ryan Cook: I completely agree. Even with the newer projects, I appreciate the community you helped build. For example, Nina Bongartz, a Red Hat

developer on the Trusted Artifact Signer Team, built an AI model verification operator with Sigstore. The fact that there is one community looking out in so many different places is a testament to what you helped produce.

THE NEXT BIG CATALYST

Ryan Cook: You see so many startup projects in this space, and sometimes they put a strange license in place allowing you to partially use the product but you can't go enterprise with it. With Stacklok and even on your newest project AgentUp, why did you decide to start as open source? You could have locked those down like some other projects out there and tried to go for gold. Instead, you're doing things to benefit the greater community, developers, and people getting started. How did you make that decision?

Luke Hinds: Good question. First, I'm doing what I've always done and what I know. But I've always found that open source comes with its benefits. You have a big audience to validate and test against, and you never come across as spamming people to get them to use your new product. You get that early diversity in as well: other people can tell you what they think, and if it's good then generally other people start to contribute and start to use it more. It's a really good litmus test for a new project. If it's not good, it's not going to grow. I'd like to say it was only for the good of humanity, but it is a little bit of a selfish move as well, because it's a great model.

Ryan Cook: There's so much joy when I get a first contribution outside of my team or from another company. I have a small celebration at my desk and I'm dancing around the room, because it gives you that validation that you're on the right path.

To wrap things up: what do you see on the horizon for security? As we've been saying, security is something many of us think about after the fact. What do you think is the next big catalyst in security for AI?

> Agent identity is a big area people are starting to approach, but so far we're retrofitting old tools and old protocols where we should take the opportunity to really rethink things.

Luke Hinds: An interesting one is agent identity. We have a lot of systems we're trying to retrofit for AI and agents, and they are creaking a bit with our current authorization approaches and protocols. There is going to be a world where an agent will need to delegate a task to another agent, with no human in the middle, and so many of our authentication systems are human-centric. They're based on a human identity.

But what about when it's not really Luke Hinds with the cute little avatar that's writing the code; it's Qwen or Claude. Agent identity is a big area people are starting to approach, but so far we're retrofitting old tools and protocols where we should probably take the opportunity to really rethink things. That's one area I've been playing around in.

Ryan Cook: I completely agree with you. We can be retrofitting what we know and trust, but doing it in an efficient manner where we're not trying to square the circle.

Luke Hinds: Coincidentally, on Sigstore we recently collaborated with someone at Red Hat on an A2A (agent to agent) store project, which was around agent identity—or agent provenance, really, so when an agent presents itself you know the code it was built from. Sigstore is quite good at retrofitting in that setting.

Ryan Cook: That is fantastic. With Sigstore, there's a level of trust people already have. You know if something comes from that organization, people approached it with the right thought process, the right review process, to make sure nobody introduces any gaping hole. Having that open community is almost a cheat code for an organization to adopt some safety mechanisms— you don't have to come up with them on your own because there's an entire field of experts out in the open helping to do that.

Luke Hinds: Absolutely. And there's some way smarter people than me active on the project now.

Ryan Cook: That's it for my questions. Thank you for taking the time—this was really fun.

Luke Hinds: Thank you, Ryan. I had fun as well.

What is Red Hat developing

NEXT?

Learn more at

next.redhat.com

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 7:2

Red Hat

Feature

GREEN.DAT.AI: an energy-efficient, AI-ready data space

Data silos, regulatory compliance, and resource consumption limit the collaboration needed to address real-world challenges. A global consortium is working to change that.

by Ben Capper

About the Author

Ben Capper is a Software Engineer at Red Hat. He is currently working on the AC3 and GREEN.DAT.AI EU Horizon research projects with a focus on green energy, AI, and cloud-edge computing.

Significant challenges have hindered the rapid integration of artificial intelligence (AI) in key industries that drive economic and social development such as agriculture, finance, and energy. Shared data can provide substantial efficiency benefits, enabling more effective and sustainable processes. However, datasets isolated due to privacy considerations, data interoperability challenges from incompatible frameworks, and data sovereignty principles under regulations like GDPR create substantial barriers to collaboration.

AI also creates energy consumption challenges. Large-scale model training is hampering efforts to reduce emissions, in turn causing delays in transitioning to green sources of energy, in direct conflict with sustainability initiatives. Compounding these issues is the absence of consistent methods for measuring, let alone reducing, the energy consumption of AI workloads.

The GREEN.DAT.AI project addresses these challenges by applying its solutions to real-world problems in four major industries, across six distinct use cases (UC):

- In the energy sector, UC1 and UC2 provide solutions to enhance efficiency and security in the renewable energy market and in EV charging.
- In agriculture, UC3 and UC4 improve resource management for farming and water management.
- In the mobility sector, UC5 demonstrates enhanced energy demand optimization and refined data analytics for an e-bike network.
- In finance, UC6 provides robust fraud detection solutions that ensure compliance.

A FRAMEWORK FOR SUSTAINABLE AI

As a solution to these challenges, the GREEN.DAT.AI project, funded by the Horizon Europe research grant program*, launched in January 2023. The project unites a consortium of 18 partner organizations from 10 EMEA countries to design and develop an energy-efficient

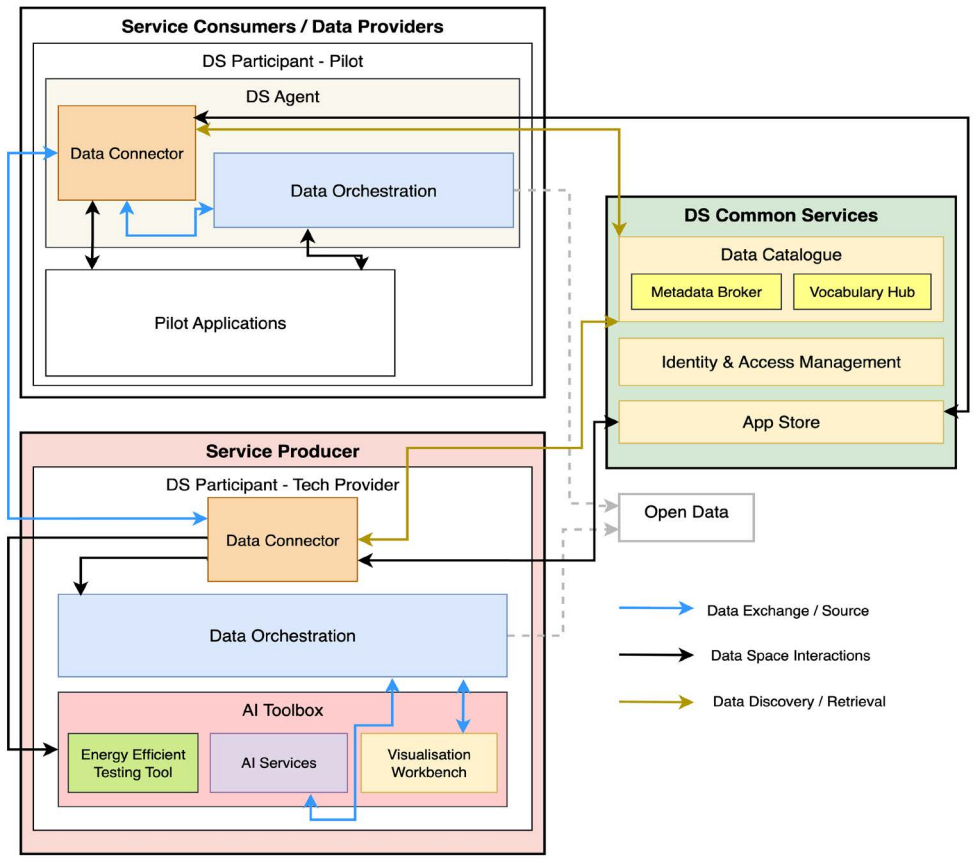


Figure 1. *Simplified GREEN.DAT.AI reference infrastructure*

AI-ready data space supported by a toolbox of AI services. These services are leveraged across various industry UCs to showcase the project's substantial benefits. By drawing on the International Data Space (IDS) framework, ecosystems such as the GREEN.DAT.AI data space ensure interoperable, GDPR-compliant data sharing that adheres to standardized exchange protocols. This reinforces data sovereignty by empowering organizations to retain control over their data within a trusted network.

The following section details the project's reference architecture, a key output of which illustrates the core components and services that bring this vision to life. We will explore how common services, the AI toolbox, and other key elements are integrated to enable efficient and secure data sharing.

REFERENCE ARCHITECTURE

A data space (DS) is a decentralized and secure data-sharing ecosystem where multiple organizations can share and access data while maintaining control and sovereignty over their own. The GREEN.DAT.AI Reference Architecture, shown in **Figure 1**, details an implementation of this project's AI-ready data space that delineates roles: Data Providers manage data ingestion; Service Producers develop and deploy optimized and energy-efficient AI tools and services; and DS Common Services facilitate data discovery, access, and interoperability through defined standards like DCAT-AP and RDF.

DS Common Services ensure adherence to the FAIR principles of data management (Findable, Accessible, Interoperable, Reusable) through a data catalog, a component that makes datasets discoverable and understandable. The catalog, implemented by Red Hat engineers, is Piveau. It provides all three required components: a data catalog with a user-friendly interface for dataset discoverability and understandability; a vocabulary hub, which provides standardized terminology for interoperability; and a metadata broker, facilitating the discovery and exchange of metadata between different data sources. The architecture also integrates secure data transfer components, including Eclipse Data Connectors (EDC) and Apache Kafka, enabling privacy-preserving exchanges and ensuring GDPR compliance within a secure, sovereign data-sharing environment.

This architecture is the foundation for the project's core innovations, which include a dedicated Energy-Efficiency Testing Tool and a comprehensive AI services toolbox. The toolbox includes advanced capabilities such as Federated Learning, Explainable AI, and AutoML, all designed to support

the development of sustainable and transparent AI solutions. The testing tool is incorporated to validate the energy efficiency of these services, ensuring they meet the project's sustainability goals.

ENERGY-EFFICIENCY TESTING TOOL

The Energy-Efficiency Testing Tool is a key part of the project's strategy to address the energy consumption challenges of AI. It provides a standardized way to measure the performance of AI algorithms from several perspectives: energy consumption, model precision, data volume, and time. By capturing these metrics, the tool can establish a baseline for an algorithm's initial performance. Subsequent iterations of the algorithm are then tested against this baseline to measure the impact of changes and validate improvements in energy efficiency.

The testing tool also includes an intelligent Co-pilot agent that uses reinforcement learning to propose new, optimized configurations to users. This agent learns from the outcomes of past tests and suggests configurations that balance high performance with low energy consumption. This automated approach helps ensure that the project's services consistently meet or exceed the goal of reducing energy consumption by 10+%.

ENERGY-EFFICIENT AI SERVICES TOOLBOX

At the core of the project is a toolbox of 21 AI services. This includes generalized services that support foundational processes, such as data-enrichment tools to improve data quality. The toolbox also provides more advanced services, such as Federated Learning frameworks that enable collaborative training on private data, Explainable AI tools that bring transparency to complex models, and AutoML services that intelligently optimize the development of new models. Collectively, these services prioritize environmental sustainability and regulatory adherence while supporting a range of industry requirements across the UCs.

Federated Learning

Federated Learning (FL) is an innovative approach to AI that allows multiple organizations or devices to collaborate on building a powerful AI model without ever sharing their raw, private data. This approach breaks down data silos and makes models more accurate for a common objective. It's particularly useful for industries with strict privacy regulations, such as banking or healthcare, but applicable for other industries as well.

For example, in Use Case 3 (UC3), farms want to train a model to optimize fertilization. Instead of sending sensitive farm data, such as soil-health tensors or nutrient sensor readings, to a central server, each farm keeps its data local. They train a version of the model on their own data and then send only their updated model, rather than the data itself, to a central coordinator. This coordinator aggregates the model updates from various farms and sends an improved global model back to each. This enables the model to learn from a massive, collective dataset while preserving the privacy of each farm's information. The project also applied FL to wind energy in UC1, where different power plants improved their energy production forecasts by collaborating without revealing their proprietary operational data. This approach directly addresses regulatory compliance rules like GDPR by keeping sensitive data on-site.

The service also has significant environmental and performance advantages. Compared to centralized model training, local model training drastically reduces the need for large-scale data transfers and energy consumption by only sharing small model updates. Results showed an optimized federated approach could consume 52% less energy than a centralized one, as seen in **Figure 2** (overleaf), demonstrating that FL is not only a privacy-friendly solution but also a key strategy for making AI more sustainable.

Explainable AI

A major barrier to the adoption of sophisticated AI models is their black-box nature. As models become more complex, it becomes nearly impossible for humans to understand how they arrive at a specific decision, which erodes trust and makes it difficult to comply with regulatory requirements. The need for transparency is especially critical in sectors like finance, where AI decisions about fraud can have serious consequences. This is the problem that Explainable AI (XAI) is designed to solve.

XAI services are designed to make complex AI models transparent and understandable. For example, in fraud detection for banking UC6, a bank needs to know not just that a transaction is fraudulent, but why the AI flagged it. This service provides tools like SHAP and LIME, which can

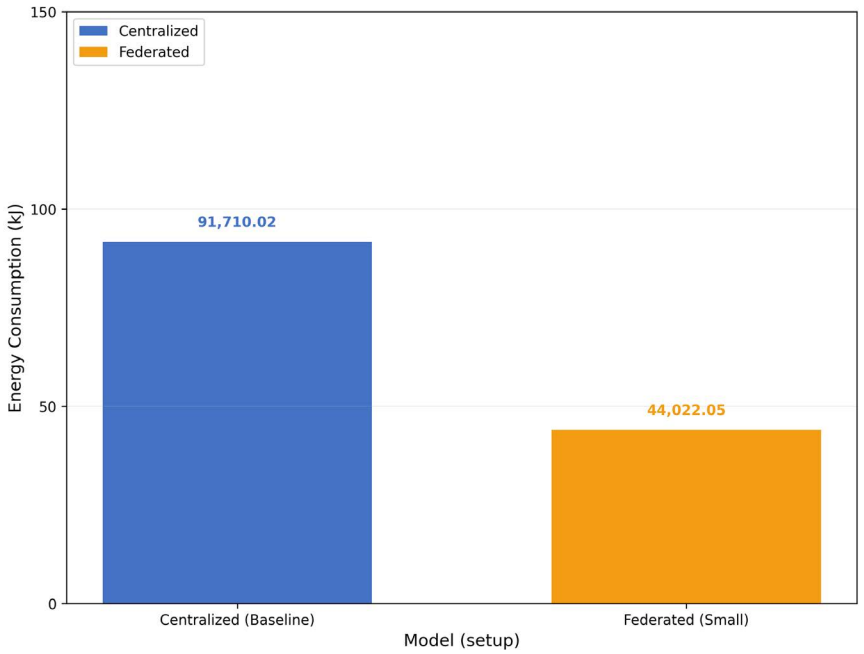


Figure 2. *Total energy consumption of centralized model (blue) vs. optimized federated model (yellow)*

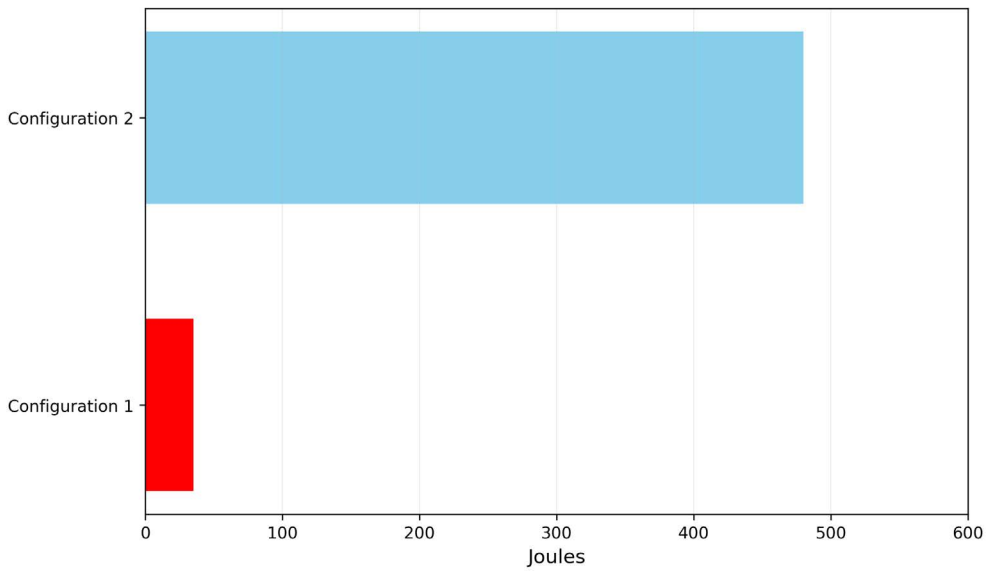


Figure 3. *Comparison of energy consumption for the maximum number of created features using feature learning (measured in Joules)*

explain an AI's decision by showing which data features had the biggest impact on the outcome. These tools build a simple, easy-to-understand model around a specific prediction to explain it, allowing users to trust the AI's judgment and understand its reasoning.

The project's toolbox includes a related service called Explainable Feature Learning, which automatically creates new, powerful features from raw data to improve an AI model's performance. Most importantly, it does so while ensuring the features are still interpretable, so experts can understand how the new features relate to the original data. This service is particularly valuable for tackling data complexity without sacrificing transparency.

The project demonstrated that these tools are not just for explanation; they can also be optimized for efficiency. An experiment with the Explainable Feature Learning service showed that by adjusting the settings to limit the number of features created, the model could achieve the same level of accuracy while consuming over 94% less energy (**Figure 3**). This shows that the process of making AI more transparent can be done in a highly energy-efficient manner.

AutoML: algorithm selection and hyperparameter tuning

Another obstacle to efficient deployment of AI is the labor-intensive, computationally expensive process of algorithm selection and hyperparameter tuning, or AutoML. This task requires trying out many different models and configurations to find the one that performs best for a specific dataset. This trial-and-

error method is time-consuming for developers, and it consumes a vast amount of energy, directly conflicting with the sustainability goals. To automate and accelerate the process, the project developed an AutoML service that uses optimization and meta-learning techniques. Instead of searching every possible combination of configurations, the service intelligently narrows options and learns from each trial to select the next most promising configuration. This drastically reduces the number of trials needed to find a high-performing model, which, in turn, saves considerable amounts of energy and time.

The service is applied to the smart mobility UC5 to address the challenge of e-bike redistribution. The goal is to predict bike demand at various stations to prevent shortages or surpluses. Using the AutoML service to quickly and efficiently find the best forecasting model for each station, the system can provide an optimal plan for moving bikes where they are needed. In a head-to-head comparison with a traditional exhaustive search (Grid Search), the AutoML service achieved a 64% decrease in energy consumption (**Figure 4**), making it fundamentally more energy-efficient and sustainable.

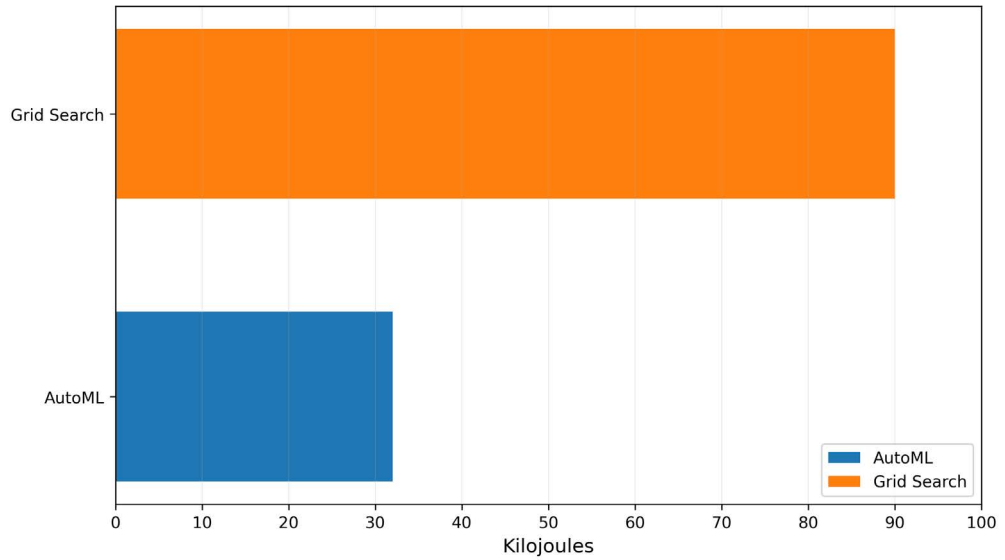


Figure 4. *Comparison of AutoML vs grid search in terms of energy consumption (measured in kilojoules)*

KEY TAKEAWAYS

Ultimately, the GREEN.DAT.AI project is a holistic, open-source-driven vision where AI is not only accurate but also sustainable, transparent, and respectful of data sovereignty. The project successfully addresses the intertwined challenges of data silos, regulatory compliance, and energy consumption in the AI era. Its holistic vision is realized through an innovative toolbox that enables sustainable and responsible AI deployment. The project's architecture, built on principles of data sovereignty, ensures interoperability and GDPR compliance through services like the EDC Connector and Data Catalog.

The Federated Learning framework breaks down data silos and allows organizations to train powerful models collaboratively without ever sharing sensitive data, an approach that supports regulatory compliance while achieving significant energy reductions. The Explainable AI and AutoML services further drive efficiency by making complex models transparent, while automating the optimization of development processes, which reduces computational waste. These services are all validated by the project's Energy-Efficiency Testing Framework, ensuring that every advancement aligns with the core objective of reducing AI's environmental footprint by at least 10%.

Concluding in December 2025, GREEN.DAT.AI project's final steps are focused on validating the results of the UCs and ensuring the technology is fully interoperable. By exchanging data with external systems and data spaces through data connectors, the project demonstrates that this solution can scale and function within a broad, interoperable ecosystem.

Red Hat engineers on the project team are Ben Capper, Leigh Griffin, Clodagh Walsh, Ant Carroll, and Ray Carroll. For more information, including documentation of all use cases, visit the project homepage (greendatai.eu), and find GREEN.DAT.AI on LinkedIn or X.

***GREEN.DAT.AI** is funded by the European Union under Grant Agreement No. 101070416.*

Funded by the European Union

THE UNIVERSAL AI SYSTEM FOR HIGHER EDUCATION AND RESEARCH

NVIDIA DGX A100

Higher education and research institutions are the pioneers of innovation, entrusted to train future academics, faculty, and researchers on emerging technologies like AI, data analytics, scientific simulation, and visualization. These technologies require powerful compute infrastructure, enabling the fastest time to scientific exploration and insights. NVIDIA® DGX™ A100 unifies all workloads with top performance, simplifies infrastructure deployment, delivers cost savings, and equips the next generation with a powerful, state-of-the art GPU infrastructure.

Learn More About **DGX** @ nvda.ws/dgx-pod
Learn More About **DGX on OpenShift** @ nvda.ws/dgx-openshift

RESEARCH QUARTERLY

VOLUME 7:2

Red Hat

Feature

Open Education Project tackles GPU scheduling and metrics visibility

About the Author

Danni Shi is a senior software engineer at Red Hat leading the development effort for the Open Education (OPE project). She is dedicated to advancing open source education and improving the accessibility of technology for all.

Enhancements to the education project highlight how research work on OPE drives advancements for many kinds of multitenant environments.

by Danni Shi

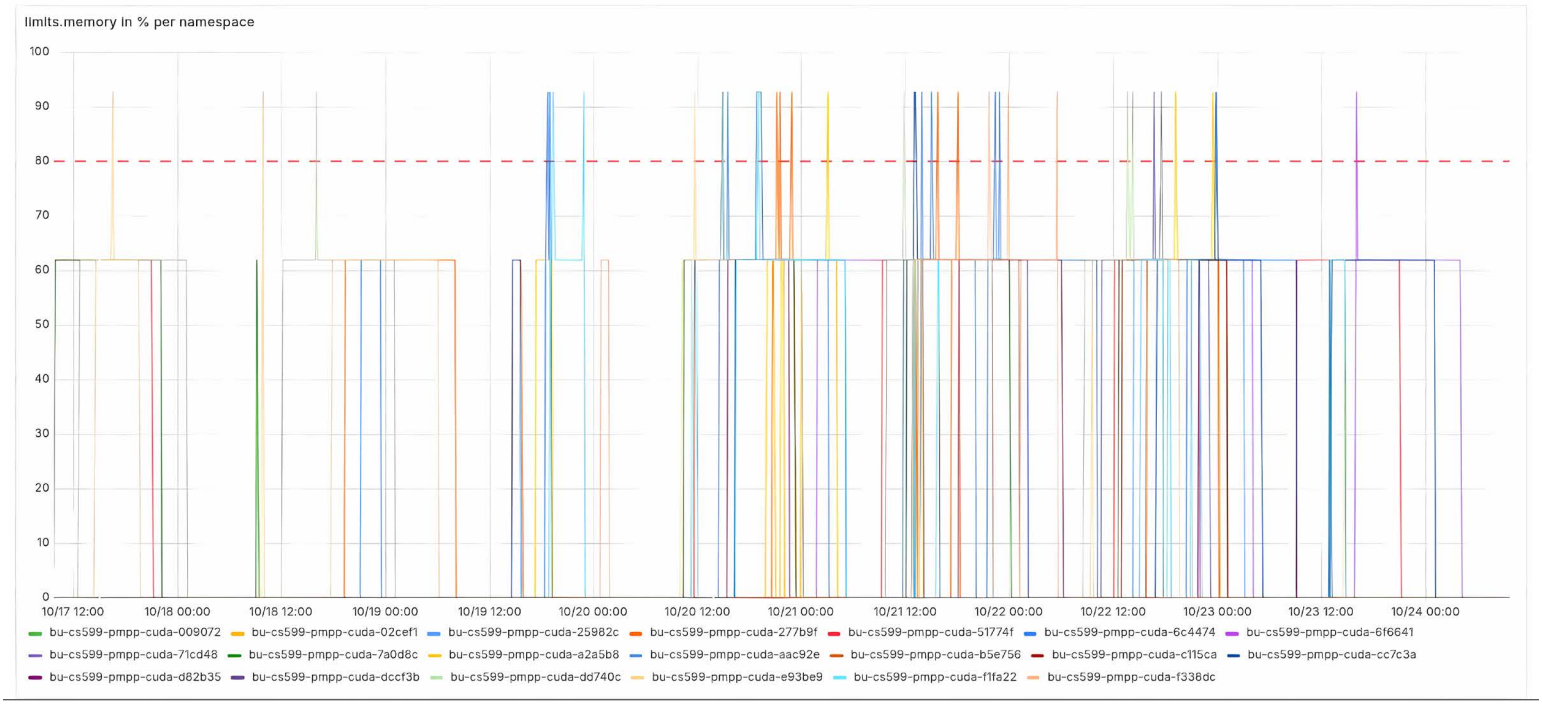
The Open Education Project (OPE) continues to develop solutions for optimizing GPU resource usage in a multitenant environment. OPE, a project of Red Hat Collaboratory at Boston University, has long been a pioneer in making high-quality, open source education accessible to all. The project was initiated by BU professor Jonathan Appavoo, in partnership with Red Hat Research, to break down resource barriers by making scalable, cost-effective infrastructure available for education.

Continued efforts to improve OPE don't benefit just students and faculty, however. Advancements in telemetry, observability, and GPU management also have potential applications for non-class use cases, such as academic research and multitenant environments more generally. Among the most exciting developments is the added ability to manage requests for processing resources. The scheduler in multitenant environments is often a black box hidden from most individual users, which means jobs won't necessarily be scheduled to keep down costs, maintain performance, or ensure that all users have fair access. By contrast, OPE-developed GPU scheduling builds in transparency and puts more control in the hands of users.

LAUNCHING GPU-ACCELERATED LEARNING

Over time, OPE has grown from an experimental class platform into a robust service supporting thousands of students. One of its biggest milestones this year has been the migration from a shared teaching and research environment to a dedicated, independent Red Hat OpenShift academic cluster on the New England Research Cloud (NERC), more widely known as the production environment of the Mass Open Cloud (MOC). This move provides the project for a sustainable, large-scale educational model.

This semester, OPE is supporting one of its most advanced courses to date: a graduate-



From the observability dashboard: memory usage per student namespace

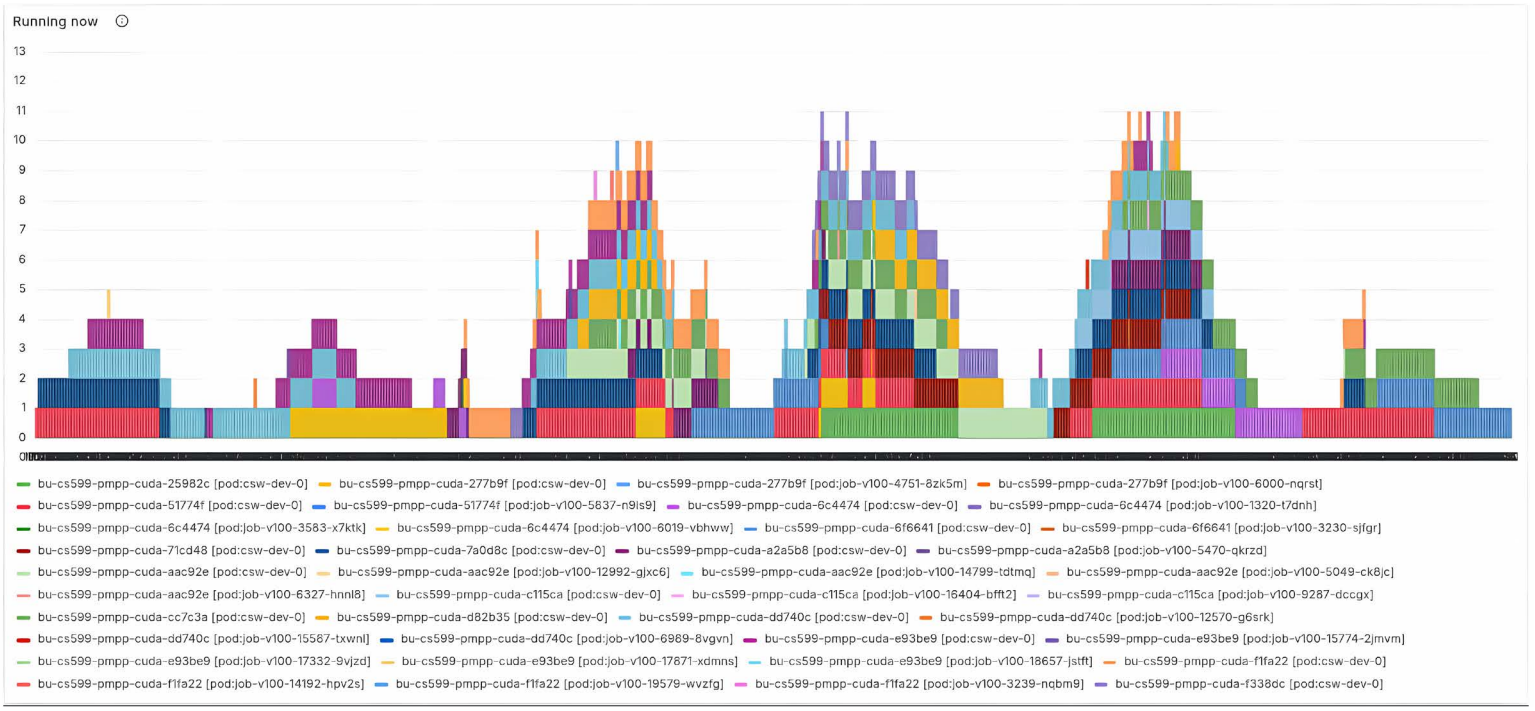
level course on GPU programming, Programming Massively Parallel Multiprocessors and Heterogeneous Systems (Understanding and programming the devices powering AI), taught by Professor Appavoo. The course introduces students to the world of GPU programming. Using NVIDIA's CUDA programming model, students learn the fundamentals of parallel programming, GPU architecture, and high-performance computing on heterogeneous systems. The course culminates in an exploration of student's research and career interests through a group project that tackles a challenge they are interested in. With interests that range from cryptographic systems to kernel engineering to RAG, students see how they can use CUDA programming to imagine more effective, performant systems.

To support this course, the team prepared a Jupyter notebook image embedded with CUDA tools. The image was specifically aligned with the NVIDIA GPU driver versions available on the academic cluster. This eliminates one of the most common pain points in computer science and AI courses—setup and installation headaches—and lets students begin experimenting with CUDA kernels, performance tuning, and architectural analysis from day one.

GPU METRICS OBSERVABILITY

Managing GPUs in an academic setting requires not just access, but also visibility. The academic cluster is connected to the NERC Observability Dashboard developed by Thorsten Schwesig. This dashboard integrates with NVIDIA's DCGM (Data Center GPU Manager) to expose essential metrics such as GPU utilization, memory usage, temperature, power draw, and per-process statistics. The dashboard also includes OpenShift metrics such as Jupyter workbench and GPU pods running per namespace, **`requests.storage`** in % per namespace, and **`limits.memory`** in % per namespace. These metrics are critical for admins to track resource consumption, not only for students but also for shared research resources.

For the GPU programming class, however, observability extends



From the observability dashboard: pods running in student namespaces

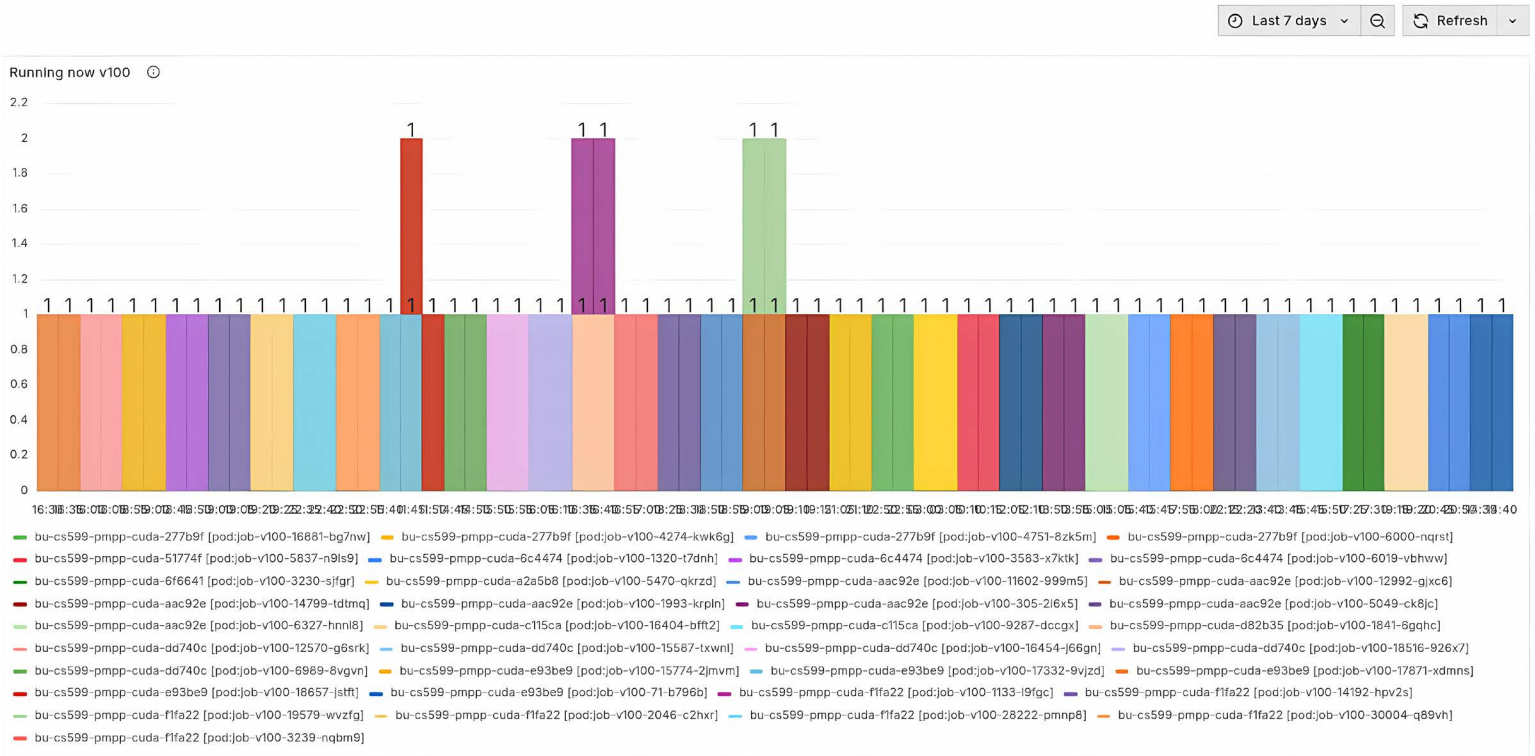
beyond operational health and aggregate datacenter resource usage. Because students are expected to evaluate the micro-architectural features of GPUs experimentally, the cluster is configured to provide access to NVIDIA Nsight Systems profiling metrics. These include deeper counters like warp execution efficiency and shared memory transactions, allowing students to connect coding decisions directly to hardware behavior. Understanding hardware-software co-design will be essential for students later working in disciplines from high-performance computing and data science to AI/ML engineering. Having the opportunity to get access to this deeper level of measurement and analysis is something that typically is not available to programmers in a shared GPU environment, much less to students in a production datacenter. It is absolutely critical, however, in order to match the demands of an application to a resource profile for a project that doesn't over-provision or under-provision compute, storage and network bandwidth as needs vary dynamically over time.

KUEUE: FAIR AND EFFICIENT GPU SCHEDULING

In the previous OPE teaching model, every student received a dedicated CPU-based Jupyter notebook. This is now a standard tool for data scientists, and the software developers who work with them. But it is also excellent for teaching most classes: CPU resources are plentiful, inexpensive, and can be allocated on a per-user or per-workload basis without significant cost or inefficiency. However, the same approach cannot be applied to GPUs. While simple, this model has significant drawbacks:

Costly idle time: GPUs are a premium resource, and when a student isn't actively running a computation, the GPU sits idle on a project, draining resources from the datacenter as a whole, and money from the individual project budget. This is a massive inefficiency for any academic institution or industry collaborator.

Limited control: This model offers less control for instructors and



From the observability dashboard: pods running on V100 GPUs

admins. Without a centralized way to manage and monitor resource usage, it's hard to ensure fair access for everyone, especially when a project has a limited number of high-end GPUs, such as Nvidia H100s.

To solve this conundrum, the OPE team adopted a queue-based scheduling model using Kueue, a Kubernetes-native job management system. Instead of tying GPUs directly to student notebooks, students now use a lightweight, CPU-based Jupyter notebook for coding and development. When they need to run their GPU code, they simply submit it as a batch job from their notebook's terminal, using simple commands.

This is where Kueue comes in. Here's how it works:

Job Queues: Each student namespace has a LocalQueue that points to a ClusterQueue, which represents the overall pool of GPUs (Nvidia V100, A100, H100) available in the academic cluster.

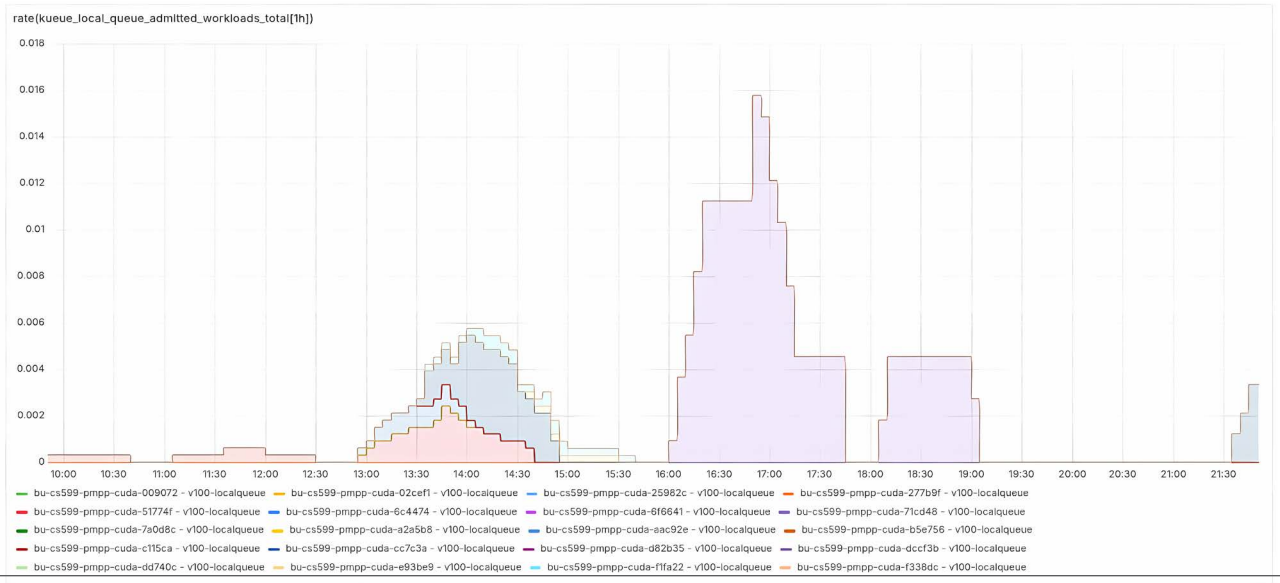
Job Execution: When a GPU becomes free, Kueue assigns it to the next pending job from the LocalQueue, triggering the creation of a pod on the GPU node. The job securely copies the necessary code files and data from the student's development notebook to the newly created GPU pod.

This creates a seamless process for coding and experimentation.

Termination and Release: Once the job completes, or if it runs for a set maximum time, the pod is automatically terminated. This releases the GPU, making it immediately available for the next student's job in the queue.

Visibility: Students can track job status directly from their notebooks (Pending -> Running -> Completed), giving them visibility and independence without requiring dedicated GPU containers.

This model ensures that GPUs are continuously utilized, fairly shared



Kueue metrics example: kueue_local_queue_admitted_workloads_total

across students, and centrally managed, making resource-intensive classes both scalable and cost-effective.

EXPANDING USE CASES BEYOND THE CLASSROOM

Beyond its use in the classroom, the Kueue system for GPU batch job scheduling offers a powerful solution for a wide range of developer use cases in multitenant environments:

Research workloads: Graduate students and faculty running long experiments can share the same GPU pool without blocking each other. A bioinformatics lab could use this to run multiple simulations simultaneously, or a physics department could manage large-scale data analysis for different research groups efficiently. The same applies to different workloads in a consulting company with many individual ML projects to complete for different applications.

Multitenant usage: Different projects can each have their own LocalQueue while still pulling from the same ClusterQueue. This benefits large organizations with various departments or external clients, such as a cloud provider offering GPU access to multiple customers or a university managing resources for different research centers.

Policy flexibility: Admins can set different quotas or priorities. For example, a research project might be allocated more A100s, while a class queue is optimized for quick turnaround. This allows for fine-grained control over resource distribution, ensuring critical projects receive the necessary resources while maintaining fair access for all users.

Fine-grained access control to telemetry and detailed hardware profiling: Because measurement and data analysis are an essential part of the developer's environment, the dashboards and collected views of telemetry provided in the OPE environment naturally support data analysis for multiple tenants or multiple projects with different access requirements sharing resources in an industry setting as well.

ADDITIONAL OPE ENHANCEMENTS

In addition to GPU management, the OPE team has made other key enhancements to the Mass Open Cloud environment:

Preloaded Notebooks: To streamline the learning experience, students' development notebooks are now pre-generated rather than launched manually through the UI. Meera Malhotra developed this feature to enable each notebook to be provisioned with the required LocalQueue configuration and

Boston University Professor Jonathan Appavoo teaching Programming Massively Parallel Multiprocessors and Heterogeneous Systems (Understanding and programming the devices powering AI)

associated RBAC permissions, ensuring it is immediately ready for assignments and labs. Students simply log in and run their notebook without needing to perform any additional setup. This way, students can focus on experiments instead of troubleshooting environments. The same advantage can also be available for developers in other MOC projects.

Gatekeeper policies: In other courses where students launch notebooks manually through the RHOAI UI without Kueue, a software Gatekeeper steps in to enforce defined resource limits. These policies, developed by Isaiah Stapleton, provide the following specific protections in OPE:

- Restricting students to use only the class container image.
- Limiting container size selection to instructor-defined options, so students can't overspend resources.
- Blocking student notebooks from directly consuming GPUs through the usual RHOAI interface. Instead, student developers must submit requests for computing cycles through Kueue to claim their GPU.

Notebook labeling via mutating webhook: OPE uses a custom assign-class-label mutating webhook to automatically add labels to each notebook pod, denoting which class a student belongs to (e.g., class="classname"). This serves several purposes:

- Billing Transparency: Labels allow admins of the cluster to differentiate between students from different classes running in the same namespace, enabling accurate, class-specific billing.
- Policy Enforcement: The same labels can also be used by Gatekeeper for validating policies to enforce rules at the class level, for example, restricting images, container sizes, or GPU count differently for each course. This automated labeling webhook ensures fine-grained governance in a multi-class academic environment, but can also be used for a multi-project shared industry environment.

By moving to a dedicated Academic OpenShift cluster in the Mass Open Cloud to explore new ways of conducting classes in a stable OpenShift and RHOAI environment, while introducing GPU batch scheduling with Kueue, the Open Education Project has built a smarter way to handle expensive high-performance resources. This approach allows for students to learn on hardware otherwise closed off to most classroom settings. Students learn how to write and optimize their CUDA code, how to coalesce memory accesses, and how to handle parallel programming on a scale most students don't get to experience.

The class culminates with an exploration of how GPU programming can intersect with students' research interests, equipping a new generation of future engineers and researchers with a deeper understanding of the hardware behind AI. OPE gives student developers the chance to think critically about improving the software that runs in datacenters, both for their classroom exercises and for their future responsibilities as developers in a resource- and energy-limited world.

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

MUNI
FI

Masaryk University
Faculty of Informatics

Your **research,**

projects

and **education** partner.

FI.MUNI.CZ

RESEARCH QUARTERLY

VOLUME 7:2

Red Hat

Feature

Refined Yuga analysis tool for detecting code defects in Rust improves usability

Improvements to the research-developed tool for analyzing unsafe Rust have rendered it much more precise.

by Anne Mulhern

About the Author

Dr. Mulhern, Red Hat Principal Software Engineer, is the technical lead of the Stratis project, which is written primarily in Rust.

Yuga is a static software analysis tool for identifying lifetime annotation code defects in Rust code. At the time it was first presented in a previous article in the Red Hat Research Quarterly, Yuga's analysis yielded an unacceptably high number of false positives in the experiments conducted. Since then, the analysis has been refined. Yuga is able to detect code defects that are not detected by competing tools, and the precision and recall of its analysis are much improved.[1]

STATIC ANALYSIS TOOLS

Just about every software developer these days makes use of some static analysis tool. (A static analysis is an analysis that is done solely by inspecting a program's source and never by actually running the program.)

I suppose that the first static analysis tool invented was a type checker. Like all type checkers after it, it occasionally rejected correct programs; that is, it was *incomplete*, because its static analysis could not determine the correctness of all correct programs. It was supposed to be *sound*, however; in other words, it would reject any program that could encounter a type error. Incompleteness is a mathematical necessity for the type checker of any but the simplest language, but unsoundness in a type checker is a bug.

We can put these properties of soundness and completeness in terms of *precision* and *recall* if we think of the type checker as simply a finder of a certain class of code defects: those that can result in type errors. Recall is the ratio of true positives to all code defects in a program. Since the type checker is required to be sound, it must find all the defects, so that number is the maximum possible value, 1. Precision is the ratio of true positives to all code defects identified by the type checker. Since the type checker is incomplete, that value is less than 1: it identifies more code defects than there actually are.

1. "Yuga: Automatically detecting lifetime annotation bugs in the Rust language," Vikram Nitin (Columbia University), Anne Mulhern (Red Hat Research), Sanjay Arora (Red Hat Research), Baishakhi Ray (Columbia University). In *(2025) ACM International Conference on the Foundations of Software Engineering (FSE)* (Trondheim, Norway); *(2024) IEEE Transactions on Software Engineering* 50(10), pp. 2602-13.

```
#[allow(dead_code)]
#[derive(Debug)]
struct Foo<'a> {
    x: *const String,
    y: &'a u32, // dummy value, so that code uses 'a
}

fn bar<'a, 'b>(arg1: &'a String, arg2: &'b String) -> Foo<'a> {
    println!("in bar, this is arg1: {}", arg1);
    Foo {
        x: arg2 as *const String,
        y: &32u32,
    }
}

fn main() {
    let v1 = "Hello1".to_string();
    let v2 = "Hello2".to_string();
    let obj = bar(&v1, &v2);
    // Uncomment the line below to force the bug to occur.
    // drop(v2);
    let v3 = "Goodbye To All That".to_string();
    println!("obj: {:?}", obj);
    unsafe {
        println!("*obj.x: {}", *obj.x);
    }
    println!("v3: {}", v3);
}
```

Figure 1. *Program listing*

Type checkers are not the only static analysis tool. Many static analysis algorithms for many languages exist. Most are not part of the compiler, but many make use of compiler infrastructure to provide data for their analyses. Most do not guarantee that either precision or recall is 1, and generally neither is. Recall can be maximized by a strategy of declaring that there are errors everywhere, but this strategy comes at the expense of precision.

In general, a good static analysis tool should have high recall: in any given program it should find most of the kind of code defects that it is designed to find. It should also have high precision: it should not identify many errors incorrectly. If a static analysis tool has low precision, a developer may spend a lot of time examining code that is not actually defective. If a static analysis tool has low recall, the code defects will persist in the program and, sooner or later, this will result in bug reports, unless the developer finds the problem by other means.

Recently, code review tools that use large language models (LLMs) to check proposed code changes for code defects have become available. In principle, recall and precision metrics for these tools could be obtained. However, unlike traditional static analysis tools, these LLM-based tools are non-deterministic. Typically, a traditional static analysis tool, for example Rust's Clippy, consists of many separate analyses that run independently and can be turned on or off as desired. Given a dataset, recall and precision values for any particular analysis can be calculated. It is the nature of LLM-based tools that their analysis is indiscriminate; they do not check for anything, they just respond. The recall and precision values will vary, even for the same dataset.

RUST

The Rust language has a flexible and expressive type system. Its type checker is conditionally sound and incomplete. There are certain Rust functions, explicitly labeled unsafe (e.g., **`std::mem::transmute`**), that allow the developer to change the type of a region of memory. If the user uses these functions in a correct way, the type checker is sound; otherwise it is not. Due to the flexibility of its type system, the Rust type checker's precision is higher than that of many other languages. Its recall value is 1 if the developer does not make incorrect use of unsafe functions that affect types.

The compiler also provides an ownership checker. This tool identifies a different kind of code defect and prevents a different kind of bug than the type checker. The bugs that the ownership checker must prevent are memory-related bugs.

Unlike, for example, Java, Rust has no garbage collection; unlike C, it does not rely on explicit memory management. Instead, it avoids common problems, like use-after-free, that plague C programs by enforcing its ownership rules. Every region of memory being used by a Rust program has just one owner. After that owner is no longer live, but never before, the region can be freed. It is the job of the compiler to emit code that will free the memory once it is unused, so that it can be reused. This must be done perfectly, in order to avoid "memory leaks."

Although a region of memory may have only one owner, there can be many borrowers. These borrowers can also access the region of memory. It is vital that no borrower access memory after the owner has ceased to be live; if it ever does that, the memory may have already been freed and reused for another purpose during program execution. If that happens, the reference will read a garbage value, which will result in undefined behavior. So, there are rules that ensure that no borrower can be live after the owner has ceased to be live. The Rust borrow checker exists to enforce these rules.

To enforce its rule that borrowers must not live longer than the owner of a region of memory, the borrow checker has a concept of lifetimes. It uses sophisticated analyses to determine whether the lifetime of a reference to some region of memory may extend beyond the lifetime of the owner. Since that could result in undefined behavior, its job is to prevent this from happening.

The Rust compiler can infer a large proportion of lifetimes without assistance, just as it can infer most types in a program. In some cases, however, the developer must use lifetime annotations to inform the compiler what lifetimes it should assume. This is something the developer can get wrong, and the consequences of an error can be a memory safety bug. Like the type checker, the borrow checker is only conditionally sound. If the developer is able to add all the lifetime annotations correctly, the borrow checker will not allow any of the memory-safety violations it is designed to prevent. But if the developer adds certain lifetime annotations incorrectly, the borrow checker may allow a memory violation.

Few developers, when they first encounter Rust, have experience developing in a language that enforces ownership rules. The errors from the borrow checker are rather confusing. Sometimes it is quite a struggle just to get the program to compile. The likelihood that a novice or even an experienced developer will fall into an error when adding lifetime annotations is high.

YUGA

Yuga is a static analysis tool to detect code defects that arise due to incorrect lifetime annotations inserted by a developer.

Incorrect lifetime annotations

Figure 1 shows a complete and buggy program where lifetime annotations have been inserted incorrectly.

```
in bar, this is arg1: Hello1
obj: Foo { x: 0x7ffcc3617cc8, y: 32 }
*obj.x: Goodby
v3: Goodbye To All That
```

Figure 2. *Possible outcome of running the program in Figure 1*

Figure 2 shows one possible outcome of running the program. **`obj.x`** is supposed to point at the string "Hello2", but when the program prints out the value at **`obj.x`**, it is the string "Goodby". What happened is that **`v2`** owned the memory containing the string "Hello2". Because of the incorrect lifetime annotation on the function **`bar()`**, the borrow checker was unable to detect the code defect in **`main()`** and the compiler calculated that **`v2`** had no live borrowers after the invocation of **`bar()`**. Hence the invocation of **`drop()`**, which consumed **`v2`** and caused the release of the memory on the heap belonging to **`v2`**, was allowed by the borrow checker.

Invoking the **`drop()`** function resulted in the memory on the heap containing "Hello2" being released and then immediately re-used for the string "Goodbye To All That". But the pointer at **`obj.x`**, the invisible to the compiler borrow of **`v2`**, remained. So when the string pointed to by **`obj.x`** was printed out, it was "Goodby", the first six characters of "Goodbye To All That", rather than "Hello2" as one would have expected just from looking at the source of **`main()`**.

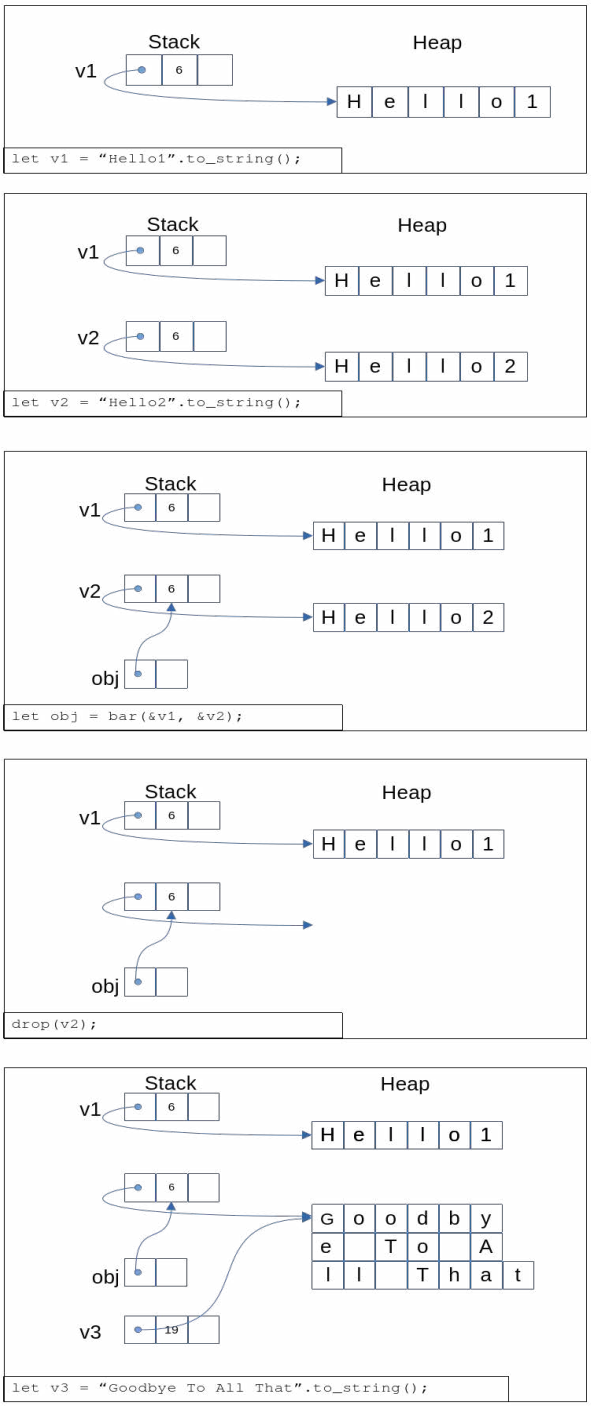


Figure 3. *The first five lines of the code example shown in Figure 1*

Several things make this example even more complex than it would otherwise be. First, while the bug could happen without the call to the **`drop()`** method, it likely will not. The borrow checker allows the **`drop()`** invocation at that point, because **`v2`** is unused after that point and it is not aware of the invisible borrow through the pointer. It would not allow a **`drop()`** invocation immediately after the assignment to **`v2`** and before the invocation of **`bar()`** because **`v2`** is still live at that point, since it must be passed as an argument to **`bar()`**.

Second, **`v2`**'s header is kept on the stack, not the heap. **`v2`**'s header, like the headers of all Strings, contains a pointer to an object on the heap that holds the characters in the string, a field indicating the size of the string, and another field irrelevant to this problem. **`obj.x`** points at the header on the stack.

Third, all that we have described so far would not have resulted in a bug without **`obj.x`** being dereferenced. Any dereference of a pointer is considered unsafe, and it must always be designated unsafe. Here, the dereference is inside an unsafe block in **`main()`**. This kind of complex reasoning is usually required with lifetime annotation bugs.

What made this bug possible was a mistake in the declaration of the function **`bar()`**. Specifically **`bar()`**'s return type is **`Foo<'a>`**, but it should be **`Foo<'b>`**. The type, **`Foo`**, is correct, but the wrong lifetime annotation was selected, and the return value's lifetime is connected to the lifetime of **`arg1`**, not of **`arg2`**. As far as the borrow checker can tell, there is nothing about **`bar()`** that requires the second parameter, **`arg2`**, to live past the print statement in **`bar()`** itself. Thus it concludes that **`v2`**'s lifetime ends with the invocation of **`bar()`**.

How could the borrow checker analyze that simple function and fail to identify the incorrect lifetime annotation? The type of x is that of a pointer to a String. The borrow checker simply ignores pointers when doing its analysis, because they are really just values, or addresses in memory. For this reason, the prudent Rust developer will avoid using pointers unless necessary. Sometimes it is absolutely necessary; Yuga could help the Rust developer go wrong less often in that case.

How does Yuga work?

Recall that the borrow checker ignores pointers when performing its analysis. Yuga's foundational principle is to pretend that the pointers are references of the kind that the

borrow checker does not ignore and to analyze the code with pointers as if the pointers were references. One rule the borrow checker uses is that if a struct contains a field that is a reference, then that field must outlive the struct that contains it. Yuga will conceptualize a pointer in a struct as if it were a reference and deduce the same rule for a pointer in its analysis, that it has an outlives relationship with its struct. Then Yuga will examine functions, using a type-based algorithm to match argument types with return types.

For the code example in Figure 1, Yuga can discover that it is possible that one of the parameters of bar() could end up in the returned value of **Foo**, simply because a value of type *String can be constructed from a reference to a String. Thus, just reasoning by types, **`arg1`** or **`arg2`** could end up in the return value; therefore Yuga reasons that to be safe, both should have lifetimes that exceed that of the return value. The borrow checker will only ensure that holds for **arg1**, since **Foo**'s lifetime parameter is the same as that of **arg1**. Therefore, if **arg2**'s value becomes part of the **Foo** return value, there is a bug.

Such a simple, type-based approach is bound to yield far too many possibly buggy functions that are not actually buggy. It is a high-recall, low-precision analysis that would seriously waste a programmer's time. Yuga refines the analysis and improves its precision by a points-to analysis on individual function bodies. The points-to analysis increases the precision, but does not reduce the recall. Yuga has experimented with several additional ways of reducing the number of false positives, with some success.

RESULTS

Yuga's performance was tested under several scenarios. First, it was tested against a well-known set of Rust lifetime annotation bugs in nine crates identified by searching the Rust Security Advisory database (rustsec.org). Second, a dataset of synthetic bugs was prepared. And finally, Yuga was run on a dataset of 375 crates on crates.io. These crates were selected from the top 2,000 crates. Crates that did not use unsafe code or lifetime annotations were removed, leaving just 375 popular crates that might have lifetime-annotation bugs. Yuga found three completely new bugs, as well as many instances of suspicious code.

Three new bugs may seem like just a few, but it is important to remember that these bugs were in crates that see a lot of use and, hopefully, get a lot of scrutiny. Less well used and examined crates would be likely to contain a greater proportion of such bugs

CONCLUSION

Yuga is a research prototype, not a production-ready tool. Yuga searches only for one kind of very difficult-to-reason-about code defect which other bug finders generally do not detect at all. We should not overlook the benefit of novelty in a bug-finding tool. By using a tool that reports a kind of defect other tools can not detect, the developer learns to understand this kind of defect and to avoid it in future, even if the tool is not always available. Most developers will put up with a high rate of false positives, so long as they are learning something new and permanently useful. That is largely the appeal of the LLM-based code review tools at this time.

The prudent Rust developer will avoid using pointers unless necessary. Sometimes it is absolutely necessary; Yuga could help the Rust developer go wrong less often in that case.

Red Hat

RESEARCH QUARTERLY

VOLUME 7:2

Column

About the Author

Taj Salawu is a software engineer at Red Hat Research working on the Mass Open Cloud. Prior to that, he was an intern with Red Hat Research, working on another open source cloud development project (Operate First). He graduated from The College of The Holy Cross in 2023, where he also played D1 soccer.

Behind the cloud: the engineering work powering the Mass Open Cloud research environment

Engineers on the Mass Open Cloud are continually developing new capabilities for the research resource. Here's how.

by Taj Salawu

For many people, "the cloud" is a very abstract entity. It's a place to store their photos and data, and they don't expect to have control over it beyond setting a password for access. In the world of academia and research, however, the cloud is not just a digital archive, it's a fundamental environment for their work. The Mass Open Cloud (MOC), which I work on, is a powerful example. The MOC gives researchers in computing, healthcare, science, and other fields access to compute resources that were previously inaccessible. Collaborating with students, faculty, and MOC staff, I've seen the needs and challenges they face and worked directly with them to develop new solutions—often solutions that give us better tools and insights to benefit all users.

One thing I've observed working with these groups is that while it is easy to develop locally on a laptop, it's not always sufficient. It's hard to test scaling, projects may not have enough resources (e.g., GPUs, CPUs, or memory), and dependency issues may arise with people running different OSes. Developing in the final staging environment instead of a laptop also allows for an easier transition from development to production. By using the MOC's Red Hat OpenShift environment for container orchestration and its OpenStack environment for virtual machines, users are empowered to do a lot more at a fraction of the cost of other infrastructure providers. Students and researchers can run compute-intensive workloads, deploy applications to a production environment, and collaborate seamlessly with not only their fellow students but also with Red Hat engineers sharing their real-world expertise—all without the need to own their own expensive hardware or match complex software/driver requirements.

Currently, the MOC provides access to FC430 and 830s for CPUs and A100s, V100s, and H100s for GPUs. All of those machines have large compute power and are used to build production OpenShift


Be bold. Be boundless.
Be a Baskin Engineer.
UC SANTA CRUZ
Baskin
Engineering
engineering.ucsc.edu

The author as a college student playing for Holy Cross (Worcester, Massachusetts), where he was team captain.

and OpenStack environments. They are also available to be leased as bare metal machines, where users can install their own operating systems. This is invaluable not just for researchers, but for industry engineers. For example, Red Hat's Emerging Technology (ET) team has also used it for distributed model training development and other AI initiatives. The MOC also provides users with preconfigured telemetry to give helpful insight into what is going on on the hardware level, for example, in terms of performance, usage, and system health. To maintain this environment, ensure adherence to best practices, and—most important—continue upgrading to stay relevant and useful, a lot of work goes on behind the cloud.

MEETING DIVERSE RESEARCH REQUIREMENTS

When I started as an intern at Red Hat Research, I was assigned to work on Operate First, which, according to its GitHub page, was focused on "open sourcing operations on community-managed clusters." Its goal was to create an environment for engineers to develop and deploy applications. Sounds awfully familiar: from my internship to coming back to Red Hat full time, there was a natural progression from a project creating community-managed clusters to working on the MOC, a collection of managed clusters for research.

As an engineer with Red Hat Research, part of my job is to explore new capabilities for the MOC. This includes efforts to improve the process of deploying new clusters, creating templates, writing runbooks for processes, and testing the use of Hypershift (Host control planes) to lower resource usage when deploying multiple clusters. This work is pivotal to many users, as their development work cannot be done in a large shared cluster, for example because of access level or specific network

configuration requirements. I'm often tasked with getting new use cases working in the current environment. For instance, we were recently asked by the Red Hat Openshift AI business unit to integrate an MOC cluster into the vLLM CI pipeline as an environment where we could deploy machine learning workloads for developers. While I was able to get it running, the users employed a platform that had not previously been deployed and tested on the MOC.

When I joined the MOC team, the environment had three clusters: Production, where most users run their workloads in dedicated namespaces with specific resource allocations; Infra, an ACM hub that manages the other clusters; and Test, an environment to test upgrades and new operators before adding them to production. Over time, we have added an observability cluster, which aggregates metrics from all managed clusters and displays them using Grafana dashboards. The observability cluster, coupled with fine-grained access control, empowers users to examine bare metal metrics, information that is often integral to their development.

We also established several bespoke test clusters, providing crucial environments for groups whose development work demands full admin privileges or specific network configurations. This summer we built an academic cluster for classes taught using the Open Education Project (OPE). This cluster is upgraded less frequently so as not to interrupt classes. These additions represent a substantial enhancement to the MOC's capabilities. They also provide a good example of how we work to address diverse user needs while enriching the research and development ecosystem.

Apart from the bespoke clusters, where the users of the cluster are given full admin access, all other clusters are managed by Openshift GitOps running on the infra cluster. This allows for the repo on GitHub to be used as the main source of truth. Outside of very minor testing on the test cluster, all changes to resources happen by creating or amending a YAML manifest in the OCP-on-NERC GitHub repo.

Much like coordinating a team on the field, addressing the challenges of engineering for the MOC requires constant communication, collaboration, and a shared effort to overcome issues and achieve a working solution.

When I first started, this was a very daunting repo to look at; however, as I got a greater understanding of OpenShift, the structure of the repo made a lot more sense. Not only does it allow tracking changes, it also permits reproducibility for the clusters themselves. With the correct infrastructure and Secrets in place, applying Kustomize for a specific cluster to a fresh OpenShift install should re-create that cluster. This also meant that post configuration of new MOC clusters could be templated, speeding up the process of deploying new clusters. Building templates was one of the first issues I worked on, and it resulted in this cluster-templating repo containing several Ansible files to create an overlay for a new cluster when provided with the correct variables. Applying the generated Kustomize file installs all the common operators and configurations shared across all MOC clusters.

COLLABORATIVE ENGINEERING TO EMPOWER MORE USERS

When I was in college, I wasn't just a computer science major, I was also a soccer captain, which has proven surprisingly useful. In my current role, I've had the opportunity to work with the MOC staff, students and teachers, and various groups within Red Hat, and each group has presented a unique set of challenges. They have different sets of requirements, from specific software versions to unique network setups, which can make a single solution for everyone impossible. Furthermore, deploying applications to OpenShift often requires extensive debugging to resolve any issues that may arise.

Fortunately, I find these challenges exciting. Much like coordinating a team on the field, addressing the challenges of engineering for the MOC requires constant communication, collaboration, and a shared effort to overcome issues and achieve a working solution.

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