SACRA

EXPERT INTERVIEW

UPDATED
10/30/2024

Towaki Takikawa, CEO and co-founder of Outerport, on the rise of DevOps for LLMs

TEAM

Jan-Erik Asplund
Co-Founder
jan@sacra.com

DISCLAIMERS

This report is for information purposes only and is not to be used or considered as an offer or the solicitation of an offer to sell or to buy or subscribe for securities or other financial instruments. Nothing in this report constitutes investment, legal, accounting or tax advice or a representation that any investment or strategy is suitable or appropriate to your individual circumstances or otherwise constitutes a personal trade recommendation to you.

This research report has been prepared solely by Sacra and should not be considered a product of any person or entity that makes such report available, if any.

Information and opinions presented in the sections of the report were obtained or derived from sources Sacra believes are reliable, but Sacra makes no representation as to their accuracy or completeness. Past performance should not be taken as an indication or guarantee of future performance, and no representation or warranty, express or implied, is made regarding future performance. Information, opinions and estimates contained in this report reflect a determination at its original date of publication by Sacra and are subject to change without notice.

Sacra accepts no liability for loss arising from the use of the material presented in this report, except that this exclusion of liability does not apply to the extent that liability arises under specific statutes or regulations applicable to Sacra. Sacra may have issued, and may in the future issue, other reports that are inconsistent with, and reach different conclusions from, the information presented in this report. Those reports reflect different assumptions, views and analytical methods of the analysts who prepared them and Sacra is under no obligation to ensure that such other reports are brought to the attention of any recipient of this report.

All rights reserved. All material presented in this report, unless specifically indicated otherwise is under copyright to Sacra. Sacra reserves any and all intellectual property rights in the report. All trademarks, service marks and logos used in this report are trademarks or service marks or registered trademarks or service marks of Sacra. Any modification, copying, displaying, distributing, transmitting, publishing, licensing, creating derivative works from, or selling any report is strictly prohibited. None of the material, nor its content, nor any copy of it, may be altered in any way, transmitted to, copied or distributed to any other party, without the prior express written permission of Sacra. Any unauthorized duplication, redistribution or disclosure of this report will result in prosecution.

Published on **Oct 30th, 2024**

Towaki Takikawa, CEO and co-founder of Outerport, on the rise of DevOps for LLMs

By **Jan-Erik Asplund**

Background

After speaking to Geoff Charles at Ramp ($295M annualized revenue in 2023) and Mike Knoop at Zapier ($310M revenue in 2023) about the challenges of putting LLMs into production, we reached out to Towaki Takikawa (ex-Nvidia), co-founder and CEO at Outerport (YC S24), to talk about machine learning operations (MLOps) in a multi-model environment.

Key points from our conversation via Sacra AI:

- **The MLOps landscape has evolved from targeting researchers who preferred simple Python APIs and quick setup (like Weights & Biases) to serving DevOps professionals who need enterprise-grade telemetry, monitoring and deployment tools that can handle high-volume production workloads**. "Previously, a big focus was getting up and started very quickly. . . The people who use Weights and Biases are more interested in the mathematics than in DevOps. . . However, as MLOps moves from research

into more serious production environments, we're returning to an era where organizations want to properly instrument their LLM inference and training pipelines."

- **LLM deployment differs fundamentally from traditional software deployment because models are massive files (often 10-20GB) that must be loaded across CPU and GPU memory, creating new technical challenges around memory management, version control, and continuous deployment that existing DevOps tools weren't built to handle**. Computer vision models previously used architectures like ResNet, which was 170 megabytes. In contrast, a small 7-billion parameter LLM today is approximately 17 gigabytes - roughly 100 times larger. . . Instead of a typical Docker container, which is typically expected to be less than 1 gigabyte, you suddenly have containers with 20-gigabyte artifacts. . . Existing infrastructure update tools like Argo, Flux, or Spinnaker aren't designed to handle extremely large files with significant startup times."
- **Building on microservices and serverless, compound AI systems merge together multiple custom AI models, LLMs and diffusion models with databases and external APIs—spawning a new generation of tools like Outerport and Modal ($32M raised) that let developers automatically deploy and run multi-LLM systems across CPU/GPU architectures without worrying about infrastructure.** "[With] compound AI systems (or agentic systems, workflows), instead of running just one AI model, these systems involve multiple AI models communicating with each other - similar to microservice architectures for AI models. . . If model A takes a minute to load, followed by model B taking another minute, the entire pipeline becomes extremely expensive. . .This technical challenge explains why we haven't seen widespread adoption of these complex, compound AI systems that go beyond simple API integration."

Interview

What does Outerport do?

Outerport helps companies deploy AI models more efficiently by solving a problem known as cold start. A significant difference with AI models today compared to the past is their size. For example, computer vision models previously used

architectures like ResNet, which was 170 megabytes. In contrast, a small 7-billion parameter (which means the model is made up of 7 billion decimal numbers- or floats) LLM model today is approximately 17 gigabytes - roughly 100 times larger. This size increase creates numerous challenges. Instead of a typical Docker container, which is typically expected to be less than 1 gigabyte, you suddenly have containers with 20-gigabyte artifacts. Many of the current challenges stem from managing these large artifacts. Our software optimizes the movement of these artifacts and integrates into existing inference pipelines without requiring the installation of a completely new inference system.

Could you provide more detail about the cold start problem?

The cold start problem occurs when loading a model from disk into a machine. The process requires first loading the model from disk (or a remote object storage) into CPU memory and then transferring it into GPU memory, which involves moving large amounts of data. Even for a relatively small 15-gigabyte large language model, this loading process can take up to a minute.

The issue is particularly pronounced when working with cloud GPUs, where the bandwidth between storage and CPU memory is often limited - you're not going to be working with state-of-the-art SSDs with super-fast transfer speeds.

As a result, end users must absorb the significant cost of transferring the model from storage into CPU memory. What we provide instead is a system to manage multiple model weights in CPU memory, allowing users to quickly swap different models in and out of their GPU- with advanced tricks like page locking and parallel processing to speed things up even further.

Are there specific types of architecture, companies, or use cases where the cold start problem is particularly acute?

Any application that's latency-sensitive requires careful consideration of cold starts. For example, when serving your LLMs to customers, you might want to reduce cold starts to deliver requests faster.

There's another important use case in development regarding compound AI systems (or agentic systems, workflows). Instead of running just one AI model, these systems involve multiple AI models communicating with each other - similar to microservice architectures for AI models. These systems might include databases, external APIs, LLMs, and diffusion models all working together.

When putting these components into a workflow, you face a critical decision: either run these on separate GPU machines, where your minimum GPU count scales with the number of models, or run everything on a single machine. With a single machine, the cold start issue becomes significant because each model in the pipeline needs loading time. If model A takes a minute to load, followed by model B taking another minute, the entire pipeline becomes extremely expensive as you move through multiple loading states. This technical challenge explains why we haven't seen widespread adoption of these complex, compound AI systems that go beyond simple API integration.

What does it look like to solve the cold start problem both before and after Outerport?

Prior to Outerport, people typically loaded their models from object storage or local storage, which left many problems unresolved. While some attempted to solve these issues by writing a single script to load all models beforehand, this created new challenges. Such a monolithic script requires aligning all dependencies to load different models into one place.

Instead, we wanted to create a microservice architecture with multiple models that can coordinate with each other in a distributed fashion. Outerport runs as a daemon process on your machine, managing the models within its own container. Communication occurs via gRPC, and the process handles all memory management (storage, CPU and GPU) automatically. The API itself is straightforward - rather than using traditional Python commands like torch.load or safetensors.load, you simply use outerport.load, which communicates with the daemon to load the model. It can support multiple GPUs allocated to multiple containers on a node.

Can you talk about your early customers and where your message has resonated the most?

The work is primarily happening with companies developing diffusion models for image generation right now. The diffusion model community has strongly embraced compound AI concepts, with ComfyUI being the prime example.

ComfyUI is an open-source project that allows users to chain together node graphs of AI models. Since customizing diffusion models is straightforward and provides quick visual results, people are creating creative applications. For instance, developers take screenshots from Nintendo 64 games to fine-tune models that can reproduce the N64 aesthetic.

ComfyUI enables users to chain multiple AI models together in creative ways. A pipeline might start with generating a Nintendo 64-style image, connect it to an image segmentation model that extracts objects of interest, and then combine these elements with different backgrounds- which are generated by yet another AI model.

These workflows typically involve multiple AI models, which makes model hot-swapping times increasingly important for users. While similar workflows with Large Language Models (LLMs) are expected to develop over time, they haven't yet reached the scale seen in diffusion model communities.

Are customers excited about Outerport because it saves them money, or is it more about managing complexity and improving the developer experience?

I would say it's much more about saving money, which also happens to provide better end-user experience. When you set up a node graph in Comfy UI, waiting one minute for a generation to finish versus ten seconds simply because your models loaded faster makes a significant difference.

This faster processing allows artists to make much quicker iterations when using Comfy UI pipelines. The speed advantage not only reduces GPU hours and associated costs but also greatly improves the overall user experience. Latency and cost are intertwined.

From a developer experience standpoint, when building these tools and testing them on a local machine, the processing time can be extremely long. Is the developer experience an important consideration in this context?

That is definitely an important aspect because in machine learning development, people often use Jupyter Notebooks as an ad-hoc system for managing model weights.

One of the main reasons for using Jupyter Notebook is that you can run code in individual cells and maintain that state while continuing to program in separate cells. This allows you to separate memory management from your actual development.

However, this creates a problematic pipeline because if you want to write a different program that uses that same memory state, you cannot do that with Jupyter Notebook. Additionally, Jupyter Notebook has issues with version control and synchronizing with platforms like GitHub.

What we've done is extract the model loading component entirely from that system. A different process manages it for you, making the state persistent across multiple subsequent process runs. This allows you to maintain state and use it much more efficiently, similar to how you would in a Jupyter Notebook, but without the technical debt associated with notebooks.

Why do people use custom models?

Cost is a significant factor sinceAPI expenses can accumulate. Quality is another crucial consideration - when building an agent pipeline by connecting various APIs, you might discover it doesn't effectively perform the intended automation at the required precision. In such cases, your options are limited. You can attempt to improve the prompts or provide better context by retrieving more relevant documents, but if the model lacks knowledge or capabilities about the specific task, there's little you can do to resolve this limitation.

While OpenAI provides fine-tuning options, users still have limited control. Recent research in model steering explores manually tuning AI model parameters to achieve specific behavioral outcomes. However, this requires full access to the

models rather than using closed black box systems. These quality and cost considerations make custom models an attractive option.
A recent analysis of enterprise LLM provider usage revealed interesting patterns. While consumer space is dominated by ChatGPT and Claude, enterprise adoption shows a different distribution. ChatGPT and Claude combined account for only 34% of usage. Google's Gemini, Azure AI, and Amazon Bedrock comprise approximately 42% of the market.

The remaining 20% consists of Cohere, Mistral, and Meta's LLaMA - representing open-source and on-premises providers. Despite being a new field seemingly dominated by ChatGPT and Claude, most enterprise adoption comes from either enterprise cloud providers or open-source/on-premises models, suggesting a compelling indicator for future adoption patterns in this space.

Can you discuss custom models in terms of security, privacy, and on-premises deployment? Is Outerport part of that ecosystem?

As a deployment platform, we need to address security concerns such as end-to-end encryption of model weights. There's also an emerging field called confidential computing that's particularly relevant. In simple terms, confidential computing uses a public key architecture for hardware - you can encrypt your model so only specific GPUs can unlock it and run it in a secure environment.

This means you can restrict model usage to designated hardware components. The benefit is that even if someone intercepts your model during transport, they cannot use it without the private key embedded in the hardware.

Confidential computing is especially valuable when running models on cloud infrastructure, as it operates in an encrypted memory state. This prevents even those with access to the cloud infrastructure from reading the model's operations.

These security features will be particularly important for industries with high security requirements. The challenge lies in implementing this end-to-end approach, from encrypted storage through to deployment. While this isn't our current focus, confidential computing capability already exists in the

latest NVIDIA GPUs, and its adoption will likely expand significantly over the next 5-10 years.

One of the things you mentioned is that this kind of problem might not be as acute for SaaS companies using AI. At Ramp, they use multiple models - some smaller local models, custom models, and GPT-4 for specific problems. They optimize based on latency, cost, quality of output, and the type of output they want to generate. Is this kind of setup well suited for Outerport, or is it just that Ramp is more sophisticated than your average SaaS company using AI? Is this a different class of problem altogether?

While I'm not entirely familiar with Ramp's behind-the-scenes operations, their system appears to be something that could benefit from Outerport.

Based on what I've seen, most companies don't yet fully understand the potential use cases for AI models. Obviously, various startups are building workflows that involve complex AI pipelines - beyond simple OpenAI API calls or chat interfaces - by connecting multiple APIs and tools to accomplish tasks that previously required human intervention. However, these sophisticated implementations aren't widely developed in most companies yet.

The need for specific infrastructure tooling hasn't been completely realized, but this will likely change rapidly as organizations begin to understand how to establish these complicated pipelines.

To the extent that this is a problem that Outerport is solving, why did you choose to address this particular challenge among many others? What does solving this problem enable you to accomplish for your customers?

A significant reason stems from my experience developing apps using GPU infrastructure tools. These MLOps and ML deployment tools allow you to deploy an API using your model. However, when deploying these models, unless you have constant usage of your service - which is uncommon in enterprise settings - you face certain challenges. Many internal tools run only once or twice a day.

Even in limited-use scenarios, these tools provide value. If a task takes one person two hours to complete, and two people in the company perform it daily, that's four hours saved per day. This translates to 20 hours saved per week. Depending on employee compensation, implementing this pipeline could result in significant cost savings.

The main issue arises after deployment. Users often experience substantial latency issues - it might take a full minute just to initialize the pipeline before execution begins. From both a user experience standpoint and a practical demo perspective, this one-minute waiting period is problematic, especially when presenting to prospective clients.

Why is this a good wedge problem to solve in terms of helping solve other MLOps-type problems?

The central data structure for MLOps, especially moving forward, is models.

This requires optimizing the entire pipeline from model training, through checkpointing, to sending models to a central registry. Organizations may need to run expensive offline evaluations for compliance reasons before implementing continuous deployment.

AI models specifically need tailored solutions because they're fundamentally different from lightweight code or Docker containers, for which solutions already exist. These models represent a new type of data format that is exceptionally large, demanding their own unique solutions and deployment strategies.

Could you map out how you see the MLOps landscape?

Let me provide some context on how things have changed over time.

There are roughly three stages in the lifecycle of an AI model. The first stage is training, where you take a dataset and a model initialized with random numbers, then train that model.

The second stage is deployment, where you put this model on a GPU or CPU server and run it.

The third stage is fine-tuning, which typically happens last - after you deploy a model, you improve it further using live data collected from deployment.

This setup looked very different in the pre-foundation model era when model sizes were small enough that companies could feasibly train models from scratch.

During the training phase, companies could train their own models overnight from a relatively small dataset of 1,000-10,000 images to get reasonable results. Inference could run easily on a GPU server or even a CPU server because the model sizes were manageable.

That era was characterized by specialized models for specific tasks. When I worked on computer vision for autonomous driving, we wouldn't have one model driving the entire car. Instead, we'd have separate models for detecting objects, determining which pixels belonged to objects, and so on. These models could be loaded into GPU memory or swapped easily due to their small size.

Tooling from this era focused on training (like dataset management or monitoring jobs) and inference on custom models. Weights and Biases for metrics, MLFlow for artifact storage, Kubernetes with KNative / KServe to serve them

Today, models are much more general purpose and training models from scratch makes less sense because it's extremely expensive.

Most companies take pre-trained models like Meta's LLaMA and deploy them, then fine-tune as needed. Much of the “training cost” has shifted to fine-tuning, though many companies opt to use API services like OpenAI or Claude instead - a significant change from the previous era where API services weren't as prominent.

This shift occurred because model training is expensive, and having a good training pipeline provides a competitive edge. Deployment remains costly, and best practices for deploying large models are still evolving, primarily due to their size. So most people still tend to prototype using LLM APIs.

This means that lots of “MLOps” tooling today works at an API level. Quality monitoring for LLM API calls, logging for LLM API calls. Lots of value will be created from enabling “MLOps” for ML on APIs especially for startups.

But what’s missing now, in our opinion, is tools that look like the previous era of MLOps but specialized for the current era for custom, self-hosted models. Being able to swap large files, integrating with a wider variety of enterprise infrastructure (due to the new-found flexibility of LLMs) and data sources, support for lots of online inference through compound AI agents. These are the sets of tools that Outerport is building.

For companies that aren't doing any training and just need to deploy and fine-tune models, there are tools like Vellum. Do you see this market segmenting based on how extensively companies apply AI? Is this distinction meaningful, or will these different approaches eventually diverge or converge?

This is an interesting question without a clear trajectory. There's going to be a lot of tooling that lets users orchestrate APIs. Many agent builder tools, as mentioned, accomplish this through features like DAG editors or graph editors - similar to how Comfy UI works for image generation - allowing users to orchestrate LLM calls and integrate different tools.

These tools will primarily target non-technical audiences. Many companies are focusing on specific business units within organizations, creating tools catered to their particular use cases.

While these solutions will be widespread, enterprise companies requiring custom-built solutions will likely adopt lower-level deployment approaches, especially when they need to maintain complete infrastructure ownership without relying on third-party APIs.

For context, the previous era of MLOps tooling primarily targeted ML researchers and engineers who typically came from data science, math, or physics backgrounds rather than pure systems or computer science.

Today, LLM use cases have grown so substantially that much of the tooling needs to handle significant volume, leading to a

return to robust Kubernetes pipelines for LLM inference. As a result, MLOps tooling is transitioning from serving researchers to serving operations professionals.

How have you seen the tooling shift as part of that change?

Previously, a big focus was getting up and started very quickly. A good example of this is Weights and Biases, which has a stronghold on researchers because it lets them write simple Python APIs to log metrics into a dashboard they can view on the web. The people who use Weights and Biases are more interested in the mathematics than in DevOps.

If you ask an ops person how to log metrics, they would tell you to set up OpenTelemetry, send metrics data into a time series database, and set up a Grafana dashboard to view those metrics in a standardized way.

However, if you tell an ML researcher the same thing, they'll respond with "What is OpenTelemetry?" They view it as just "engineering" and prefer having a simple tool they can quickly implement, which Weights and Biases provides for research purposes.

During the previous era, we saw many tools where the main value proposition was the speed at which you could get started. This had enterprise value because many companies employing these researchers are big companies like Meta and NVIDIA.

However, as MLOpsmoves from research into more serious production environments, we're returning to an era where organizations want to properly instrument their LLM inference and training pipelines. This allows them to set up robust dashboards that can handle large volumes of data without crashing and with alerting.

We're moving back toward traditional DevOps tooling rather than focusing solely on tools aimed at quick and easy setup.

When we talk about deployment and this DevOps-MLOps analogy, do you think the comparison is particularly accurate? Will we see the same categories and setup, such as version control, continuous deployment,

containerization, orchestration, iPaaS, and PaaS? Or is it more of a superficial analogy?
The analogy makes sense when comparing ML deployment to traditional web app deployment, but the challenges are quite different. While web apps involve deploying code, ML deployment deals with much larger model artifacts. The process more closely resembles deploying microservice architecture than a standard app deployment.

In microservice architectures, you have systems running multiple components that need to be updated individually without causing downtime.

Tools like Argo and Flux help manage these updates gradually rather than changing everything simultaneously, ensuring users don't notice the transitions. These updates often involve larger files than just code because you're building and shipping containers to microservice clusters.

However, similar solutions don't exist for AI models today. Deploying AI models without infrastructure disruptions or complete docker container replacements remains challenging.

The existing infrastructure update tools like Argo, Flux, or Spinnaker aren't designed to handle extremely large files with significant startup times. These lengthy startup times require more careful management when transferring infrastructure from old to new versions.

Another challenge is working with heterogeneous hardware. Models must be loaded into both CPU and GPU, meaning the initialization phase isn't limited to storage. Models need to be ready on the CPU before deployment or infrastructure migration to new deployments can begin.

These unique challenges require solutions specifically designed for ML deployment, including version control and proper CI/CD pipelines. Rather than replacing Docker, we need complementary solutions that work with Docker and Kubernetes to enable live updates of AI models running on existing infrastructure.

Is Outerport's expertise primarily focused on optimizing CPU/GPU performance for large files? And does that

optimization help solve a horizontal range of deployment problems?
We're specifically starting with CPU and GPU performance optimization and implementing a daemon process that runs on your GPU because this offers the fastest time to value. Even without a complex setup, you can benefit from it - for example, as an individual developer who doesn't want to wait for the LLM to load every time you boot up your script. It serves as a convenient tool that provides immediate value.

This approach is particularly interesting because when you begin developing version control or continuous deployment systems, you need a way to deploy onto a running server. A key component of this deployment is the distributed system daemon process that handles memory management. We started here specifically because this component enables you to handle requests from continuous deployment systems - such as swapping in new models, preparing them in CPU memory, and transferring them to GPU memory when the inference process is ready.

To accomplish any of this functionality, we need the daemon process running and managing memory and models. This serves as the centerpiece of our infrastructure, from which we can build out additional capabilities spanning from training to deployment. We've chosen to start from the deployment end and work our way backwards.

When considering your target customer, are you aiming to sell to DevOps professionals or individual engineers? In terms of go-to-market strategy, are you pursuing a self-serve model where engineers might start using it on their personal machines before bringing it into their companies? Or are you focusing on top-down enterprise sales, since larger companies currently show the most interest in this technology?

The real answer, which is admittedly boring, is that we need to do both. It has to be somewhat top-down because factors like cost ultimately affect management-level people, and we need their buy-in to sell these products to big enterprise companies.

However, if developers push back saying the product is useless or incompatible with their stack, the implementation

will fail. Therefore, we need a setup that allows us to market both to developers and to DevOps leaders such as CIOs. There's an interesting dynamic regarding MLOps. As mentioned earlier, MLOps is shifting from researchers and “AI departments” (which may have different names within companies) to broader implementation. A power struggle often exists where MLOps originated in research, but DevOps teams have begun taking ownership of certain aspects.

The responsibility for MLOps now varies by company - sometimes falling under IT and infrastructure/DevOps departments, other times under AI engineering departments. Consequently, we must navigate and communicate value to both audiences to secure buy-in from both sides.

Are the major AI companies like OpenAI and Anthropic, along with cloud providers such as AWS and Azure, potential customers for Outerport? Are they already implementing similar systems internally to serve their APIs?

Cloud providers are definitely implementing versions of this approach. However, our main advantage compared to cloud providers is being a third-party vendor, which means we're not locked into a specific cloud platform.

Regarding companies like Anthropic and OpenAI - API provider companies - we would love to work with them, especially since they support features like fine-tuning. While I'm not familiar with the exact details of their fine-tuning processes and deployment methods, these companies face general deployment challenges and have developed their own solutions.

Our priority should be establishing best practices for model deployment in a standardized way, followed by evangelism efforts to spread this approach across companies for easier adoption.

Is there a competing vision? There’s one perspective that suggests there will be a single model to rule them all, where the trajectory is that this current solution is nice but temporary—more of a bridge solution rather than something of long-term importance. The alternative view is

that multiple models will continue to serve different purposes.
That is definitely a possible situation, but it's difficult to believe it could happen for two main reasons.

First, AI models require scaling both data and model size to improve, which means we need to find new data sources. Currently, most companies are consuming internet content, which is increasingly AI-generated.

Research on using synthetic data to train these models has shown mixed results. Some findings suggest that continuing to train with AI model outputs doesn't necessarily improve performance - it might actually increase hallucinations and lead to weird local minimums.

The second issue concerns costs. As AI models get bigger, the cost of inference increases. In a future where huge models dominate everything, running these models will be very expensive - certainly more expensive than running smaller models. This will likely create market segmentation.

There will be use cases where people will pay significant money for extremely sophisticated models that can accurately answer any question. However, many current applications don't require such sophistication.

For example, when filling in metadata for product information, such as determining the manufacturer of a McLaren from a CSV file, language models are very effective at handling these structured or non-regular inputs. These tasks don't require an AI model with PhD-level knowledge - a smaller, more efficient model works well.

There is also something to be said about how if you’re given two models of the same size, the specialized one will always perform better at a specialized task. This matters in scenarios where you want to maximize performance and reliability; and similar to how companies hire both generalists and specialists.

Regarding smaller models, API companies are unlikely to maintain a stronghold due to cost considerations and the thriving open-source community. We will certainly have high-quality open-source AI models in the 7B parameter range and below - smaller models that can run locally on a single GPU.

Looking in the opposite direction, is there a scenario where microservices-style orchestration becomes extremely granular, with countless small custom models? In this case, would the models become so small that the size problem essentially disappears?

That could be a possible universe.

In such a scenario, we would focus primarily on the actual model deployments themselves. While we will definitely continue to compress the models further, there's an important consideration regarding computer architecture: compute power is getting faster, allowing us to handle bigger models with cheaper GPUs, but memory bandwidth isn't increasing at nearly the same rate.

The growth in computer processing speed has significantly outpaced memory speed, which creates an inherent imbalance. This fundamental computer architecture bottleneck means that figuring out how to efficiently load these models will remain a crucial need in the long term.

Are new chips like those from Cerebras—which aren't GPUs at all—potentially an existential threat to what you're doing? How do you think about that?

As a third-party vendor, we can provide software support for any type of hardware. There's a deployment consideration where hardware has some kind of memory system that needs to load the models, and we serve as a consistent interface between models and loading them into various hardware systems.

This is actually why we named our company Outerport - we act as a consistent interface to GPU infrastructure and enterprise infrastructure, including different types of custom hardware.

Another interesting use case involves edge devices, such as security cameras running with custom hardware performing image recognition. It's challenging to determine how to live-update models running on edge hardware in remote locations.

This is a use case we're focusing on extensively since we're a deployment platform aiming to deploy to any type of hardware, regardless of location.

Can you discuss the current state of open source models? Is this development fundamentally enabling the open source model ecosystem? What does the development stack look like for open source models, and are you optimistic or pessimistic about the ecosystem's future?

I have high hopes for the ecosystem. To give some examples of useful tools: vLLM is an excellent project for running inference with LLMs efficiently. It's a runtime engine that implements GPU-level optimizations for running LLM models more effectively.

vLLM has gained significant user adoption because it's relatively easy to get started with. There's a similar solution from NVIDIA called TensorRT-LLM, which offers great performance but is more complex to use compared to vLLM.

These types of projects are encouraging because they focus on running open-source AI models cost-effectively on local hardware. There is in general a thriving community of people building local LLMs in communities like /r/LocalLLaMA on Reddit.

Could you talk about what a gold standard ML ops pipeline looks like? Since you work with many companies, what does it look like when you see a company implementing ML ops correctly?

To be completely honest, I haven't seen anything super sophisticated in this space for the most part except at certain companies with lots of resources.

The main issue is that we don't have standard tooling to handle these tasks today. Most existing tools operate at the Python level - typically Python libraries that make certain processes more convenient than before. However, these tools aren't really targeting reliability or infrastructure resilience, nor are they trying to address a broad range of use cases. Lots of tools do a lot of things decently well and not enough tools do one thing really well.

This is very much an evolving ecosystem. Previously, the use cases for ML and deep learning were more limited compared to general computing platforms, such as serverless compute platforms. However, now that LLMs and generative AI models

are permeating into different applications, we're starting to see standard tooling develop around this area. Tools like vLLM demonstrate this trend, but we're still far from achieving true standardization and establishing sensible implementation practices across the industry.

Is making Outerport an open source project important for becoming a standard and winning hearts and minds?

We are considering making it open source. While it isn't open source today, making it open source would benefit the community much more. We need to strategize about this decision further, as open source has traditionally been sensitive in terms of co-existing with the commercial offering.

Can you discuss the developer experience and deployment layers, particularly the MLOps infrastructure that's emerging from cloud GPU companies? For example, companies like Vercel partnering with CoreWeave, or similar initiatives from AWS and other major cloud providers?

Some really cool projects exist here, and one example I particularly appreciate is Modal. Modal lets you mark certain parts of your Python application to run on GPU, creating a nice developer experience.

As a developer using these platforms, you don't have to think too hard about where or on what hardware it runs - you can simply mark it as GPU, and the parts that don't run locally will execute on the cloud seamlessly.

This approach works well for smaller projects, but we'll likely see more solutions that can run on your own hardware. Organizations often prefer not to use third-party cloud providers due to uncertainty about security practices. The next evolution would be achieving a Modal-like experience that works with your own hardware.

This progression mirrors how we evolved from monolithic architecture to microservices, and then to serverless endpoints that can scale independently. The current popular paradigm involves working with multiple APIs, often internal ones, backed by serverless architecture. Now we're seeing developers who want to abstract away from thinking about APIs altogether - they simply want to call a Python function

that automatically becomes a serverless implementation, which is what Modal facilitates.
This new paradigm is particularly relevant for heterogeneous computing environments involving CPU, GPU, cloud resources, or more powerful CPUs elsewhere. However, significant challenges remain, particularly around data access. Local systems and cloud systems don't necessarily have access to the same data, so establishing efficient data access becomes a crucial problem. Despite these challenges, there's considerable excitement around these newer, higher-level paradigms of development.

If everything goes right for Outerport over the next 5 years, what does it become, and how has the world changed as a result?

Our mission is to drive wider enterprise adoption of generative AI by providing the infrastructure and tooling needed to implement it in a resilient and bulletproof way. One way to understand this is that we want to transform every company into an API. It's interesting because what a company is, at its core, is that it is a legal entity - in Japanese, this translates to 法人 or "legal person," which reflects its status as an entity with rights similar to those of an individual. So why can’t we interact with them in the same way we would as a human?

If we consider enterprise infrastructure as the brain of the company and their data as their memory, we need to enable interactions with companies in a human-like way.

Previously, the API interface to companies was essentially humans - you would walk into a bank and talk to a person to interact with that institution. This evolved into machines like ATMs, then into SaaS solutions and web banking. Now we're coming full circle, potentially creating autonomous units that people interact with through AI-powered human-like interfaces… or before we get there, through Python functions.

However, this transformation requires significant work. The infrastructure needs to be both buildable and cost-effective. There are numerous challenges, particularly in connecting enterprise infrastructure with LLM infrastructure. This is where traditional approaches like microservice architecture have proven useful, and we need to extend these principles to AI model deployments and infrastructure in general.

Disclaimers

This transcript is for information purposes only and does not constitute advice of any type or trade recommendation and should not form the basis of any investment decision. Sacra accepts no liability for the transcript or for any errors, omissions or inaccuracies in respect of it. The views of the experts expressed in the transcript are those of the experts and they are not endorsed by, nor do they represent the opinion of Sacra. Sacra reserves all copyright, intellectual property rights in the transcript. Any modification, copying, displaying, distributing, transmitting, publishing, licensing, creating derivative works from, or selling any transcript is strictly prohibited.