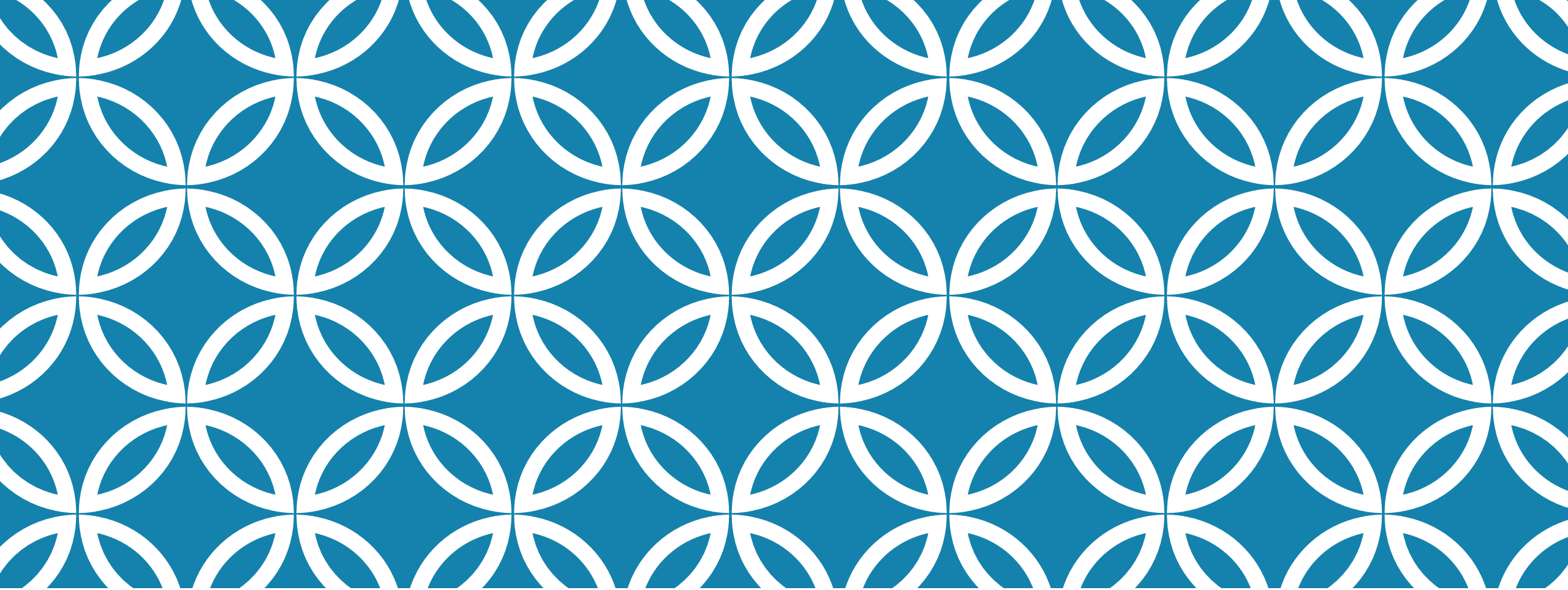


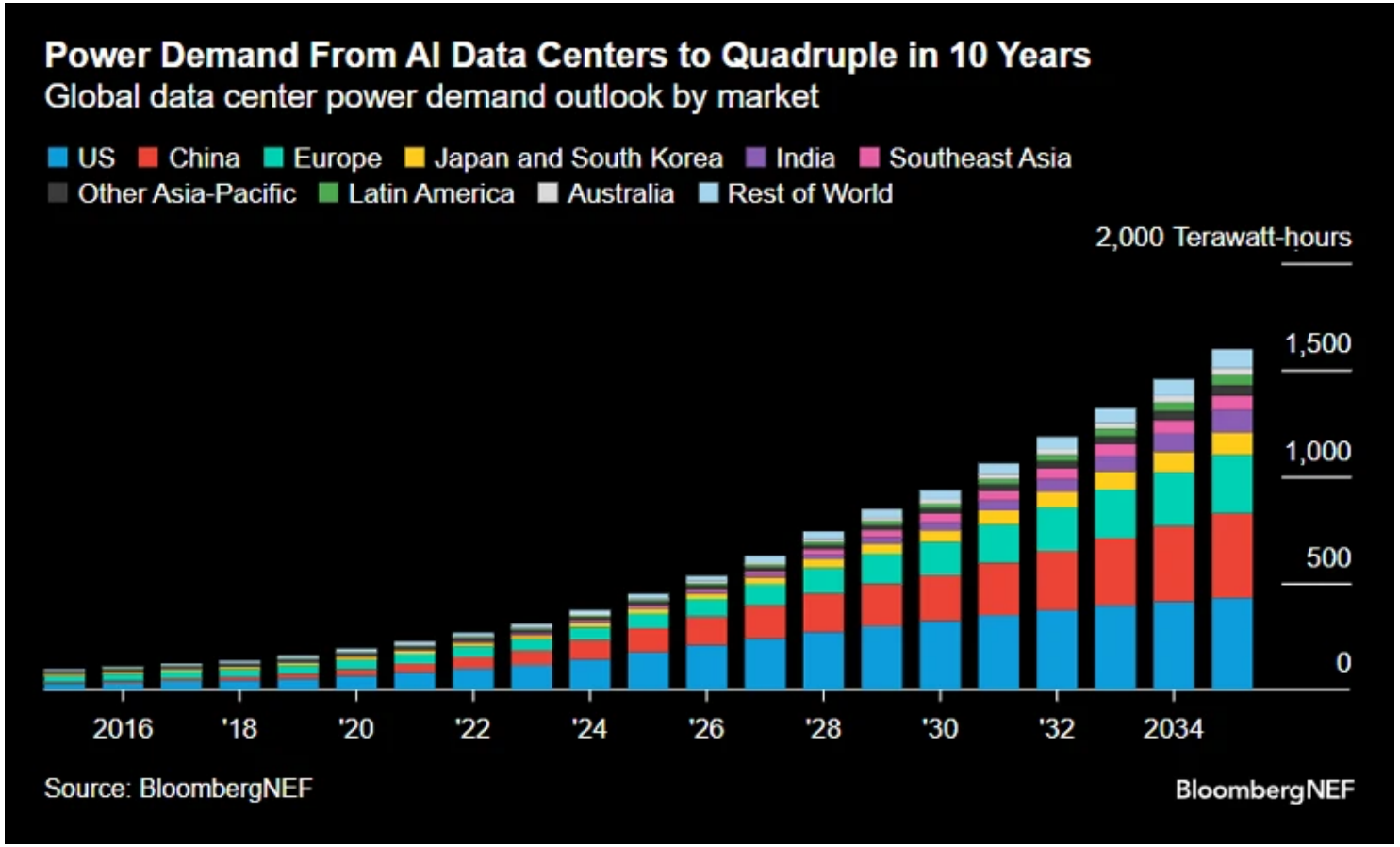
UNDERSTANDING AND IMPROVING THE PERFORMANCE OF ML SYSTEMS

Professor Ken Birman

CS4414/5416 Lecture 27

ML IS TOO EXPENSIVE

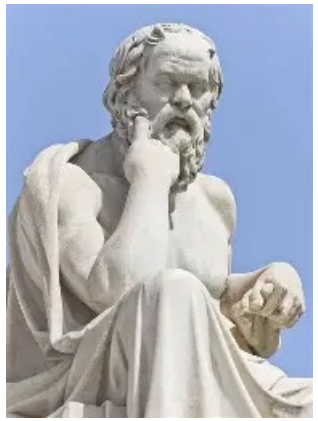
So, what can we do about that?



PROFOUND LIFE LESSONS

All I Really Need to Know I Learned in Kindergarten (Robert Fulghum)

An unexamined life is not worth living (Socrates)

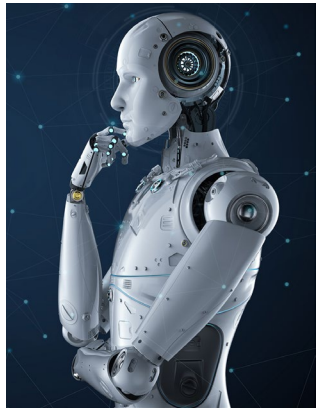


Suspect each moment, for it is a thief, tiptoeing away with more than it brings. (John Updike)

PROFOUND PERFORMANCE LESSONS

All I Really Need to Know about (ML) performance I Learned at Cornell

An unexamined program is probably very inefficient



We tend not to even look at ML systems from a performance perspective

CONSIDER TRAINING OR FINE-TUNING FOR A MIXTURE-OF-EXPERTS LRM

Where are the costs?

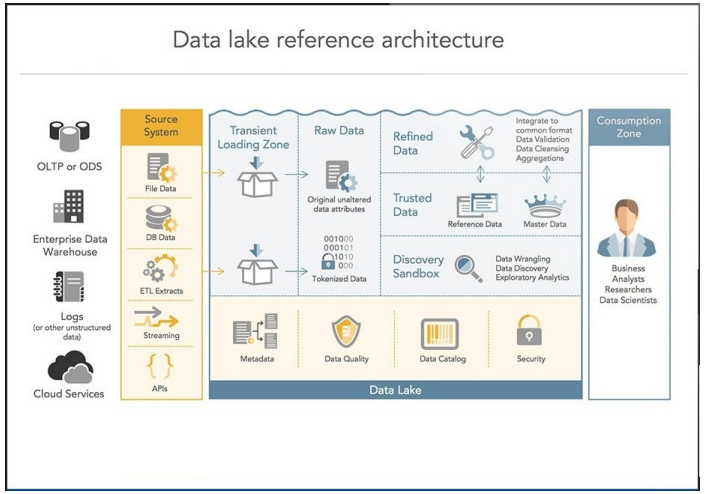
How does the ML developer perceive this question?

EXAMPLE SCENARIO

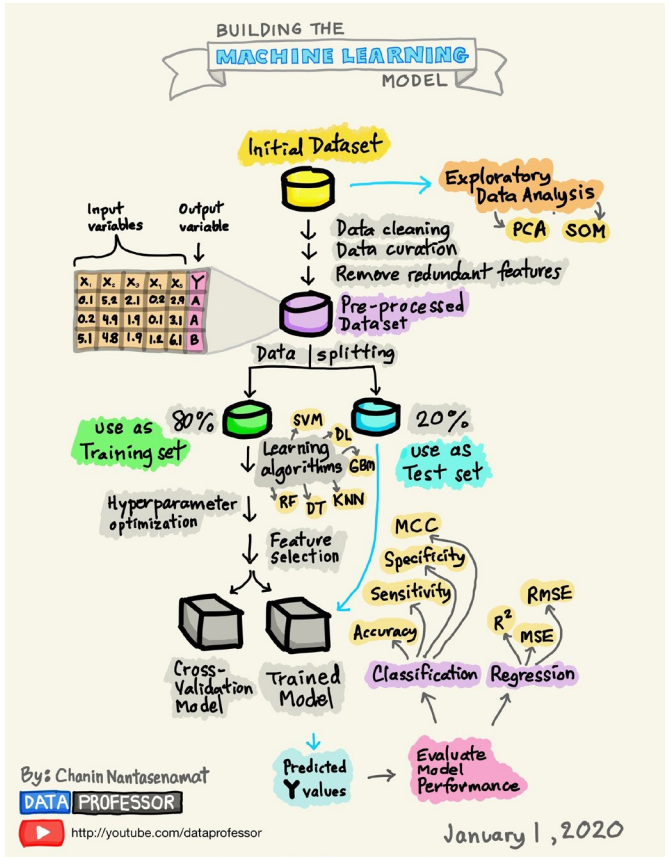
You are hired by the real estate unit of Smart Bank. It specializes in derivatives and other products tied to mortgages.

Your new job is to create an LRM distilling those years of experience into one easily-queried specialized AI trading tool.

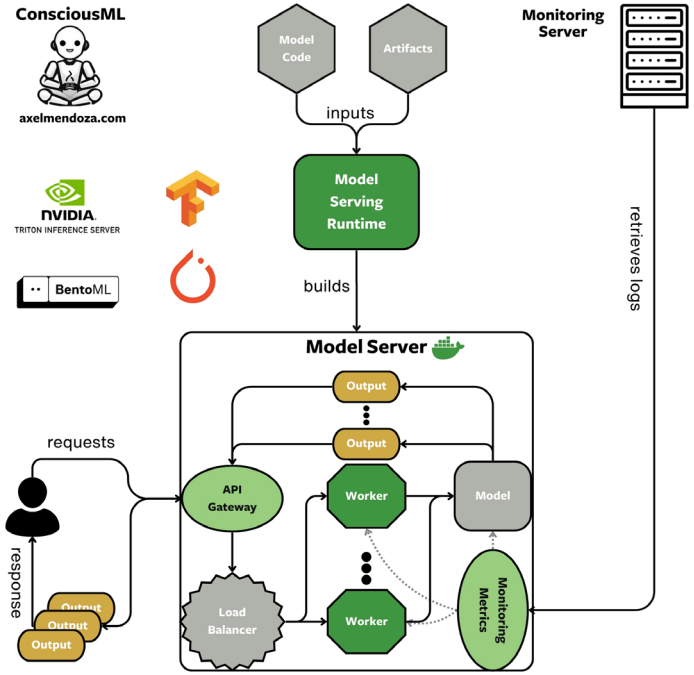
THE THOUSAND FOOT PICTURE



Data Mining to extract ML training data from source objects



ML model training or fine tuning



ML serving

Costs money, but Smart Bank won't earn anything yet

Generates revenue

BUT WHERE DID THE MODEL COME FROM?

Sometimes, off the shelf, but often you need to compose existing solutions and even create new (hopefully, small) models!

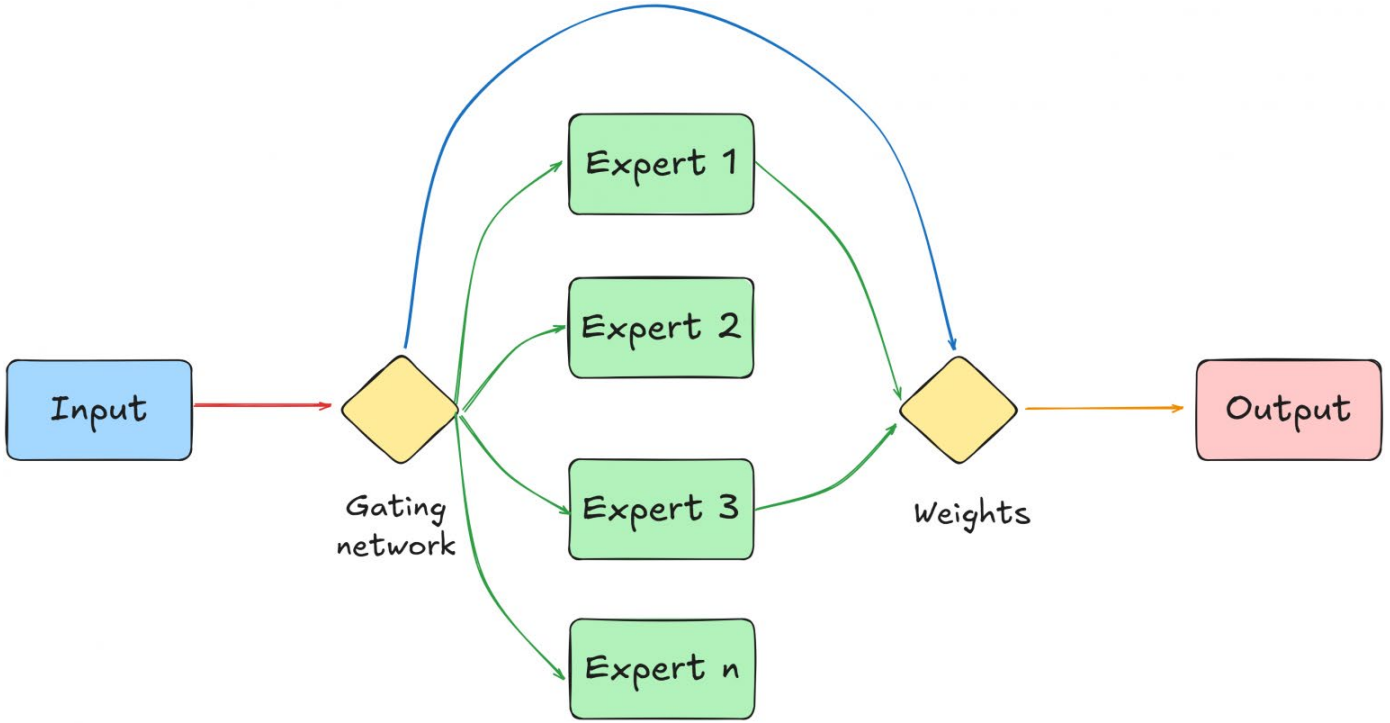
Smart Bank's mortgage-backed security trading tool probably requires a mixture of experts LRM: an MoE LRM, perhaps with an associated collection of documents: a RAG MoE LRM.

MIXTURE OF EXPERTS CONCEPT

This is an ML that has component subsystems each of which is actually a separately trained ML.

- Perhaps, an expert that knows about trading securities
- … plus one for assessing security quality based on underlying mortgages and the associated properties
- … and for estimating changing valuations for real estate in various markets, etc…

VISUALIZING A MIXTURE OF EXPERTS



MIXTURE OF EXPERTS CONCEPT

The idea is that each expert contributes a perspective and then using a weighting function, these internal perspectives are blended.

The gating network decides which experts to use and how to weigh their respective inputs.

There may be a lot of experts available, with most of them unused expect in specialized situations

SMART BANK PROBABLY WANTS…

… A mixture of experts that can support step-by-step reasoning

… Tapping into a variety of databases, some with access restrictions (policy, client privacy, legal/contractual firewalls etc)

… able to perform "agentic" tasks for the user, such as creating or editing slides, spreadsheets, etc

FINE TUNING CHALLENGE

Off the shelf models (foundation models) often need to be tuned, and a RAG LRM agentic MoE needs to learn how to weight the expert perspectives.

- Training and fine tuning are iterative and run on expensive systems.
- We also need to create and index the vector document database and tune the MoE to query it properly.
- We need good labeled data for each scenario where customized behavior is desired

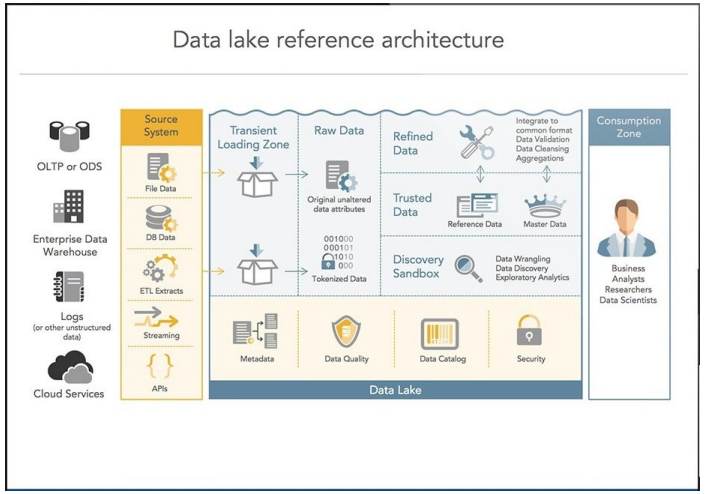
Debugging such a solution and training it takes many “cycles”.

WE UNDERSTAND SIMPLE TUNING CASES BUT THIS IS FAR MORE COMPLEX!

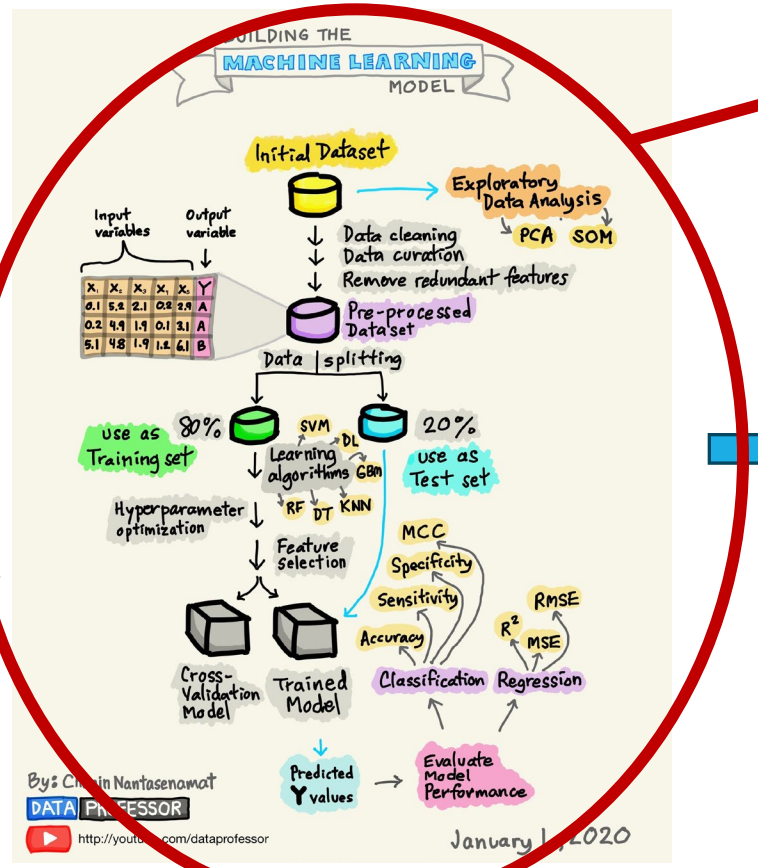
Today we train the components separately, but for this to really work they probably all need to be "co-trained" in a single run that uses RL to optimize their interactive behavior.

This kind of training will spread over multiple systems including vector databases, programs like excel or powerpoint, etc.

THE THOUSAND FOOT PICTURE

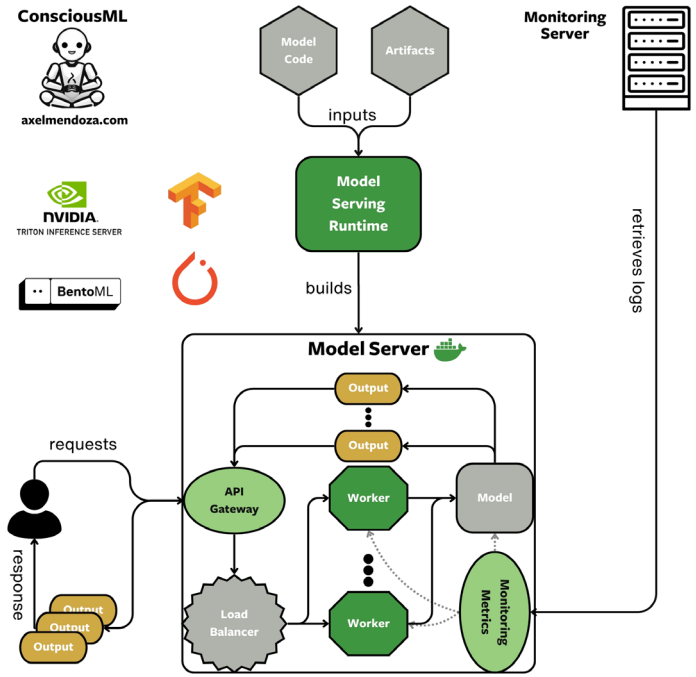


Data Mining to extract ML training data from source objects



ML model training or fine tuning

… the debugging and improvement cycles re-run this step.

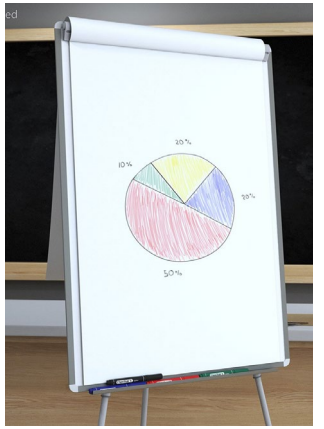


ML serving

Costs money, but you don't earn anything yet

Generates revenue

WHERE IS THE TIME / ENERGY SPENT?

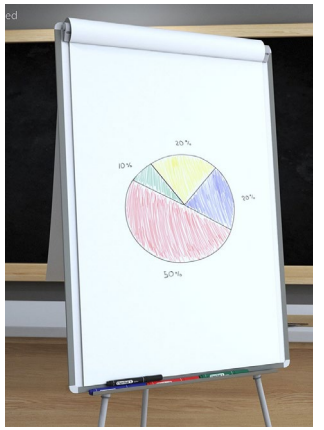


Data mining issue: the data is in multiple formats (logs, trade confirmations, prices bid and offered, etc).

Data preparation will involve

- "Data wrangling" to extract the data
- "Normalization" to get it into some common metrics, and
- "Feature engineering" to create structured training data.

WHERE IS THE TIME / ENERGY SPENT?



… this occurs at enormous scale, but the computations are cheap.

With a small amount of human feedback, automated techniques can be used to create much larger labelled data sets for training purposes.

… so, the task runs on inexpensive CPUs, securely shared with other users.

$1000!

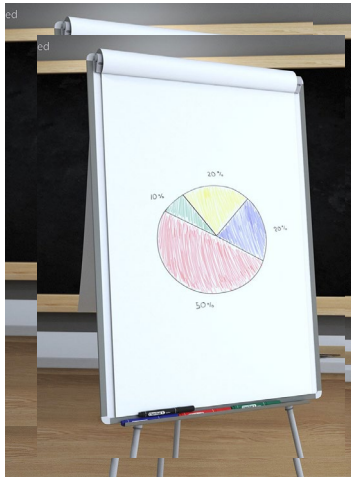
WHERE IS THE TIME / ENERGY SPENT?

Each cycle of the training task could run for hours or days.

$10K **per developer cycle** to improve the design and configuration of the training system. We'll budget 10 cycles, but this could be quite naively low… unskilled developers make errors and might need 100 or more cycles.

$100K for 10 cycles, perhaps far more

WHERE IS THE TIME / ENERGY SPENT?



Now we are finally ready to serve the model. Runs on a shared platform equipped with a few A100 GPUs

$50K for six months…

Whereas RAG LLM MoE creation is a pure cost, here the model is in use. It is "making money" for Smart Bank. Viewed as a cost of operations, not cost of R&D.

Why only six months? A model such as this will need to be "tweaked" as the market evolves, as banking laws change, etc.

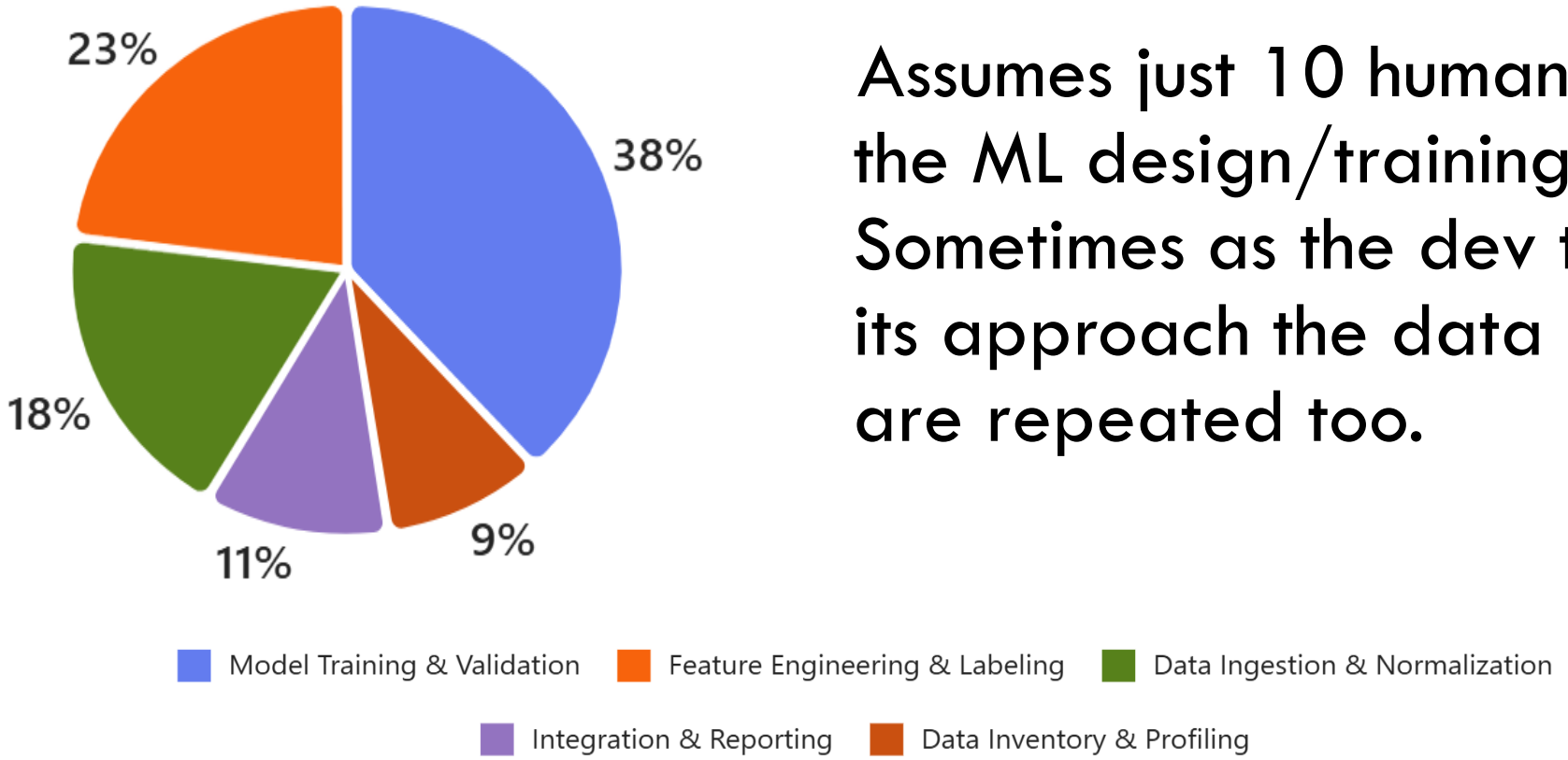
MODEL TWEAKING

In our initial SFT training set, we adjusted a foundation model to "**talk like a mortgage backed securities expert.**"

Now that we have our RAG MoE LRM, we won't need to start restart from scratch if market conditions evolve – it suffices to update the RAG database and "tweak" the language models to contextualize questions and respond knowingly.

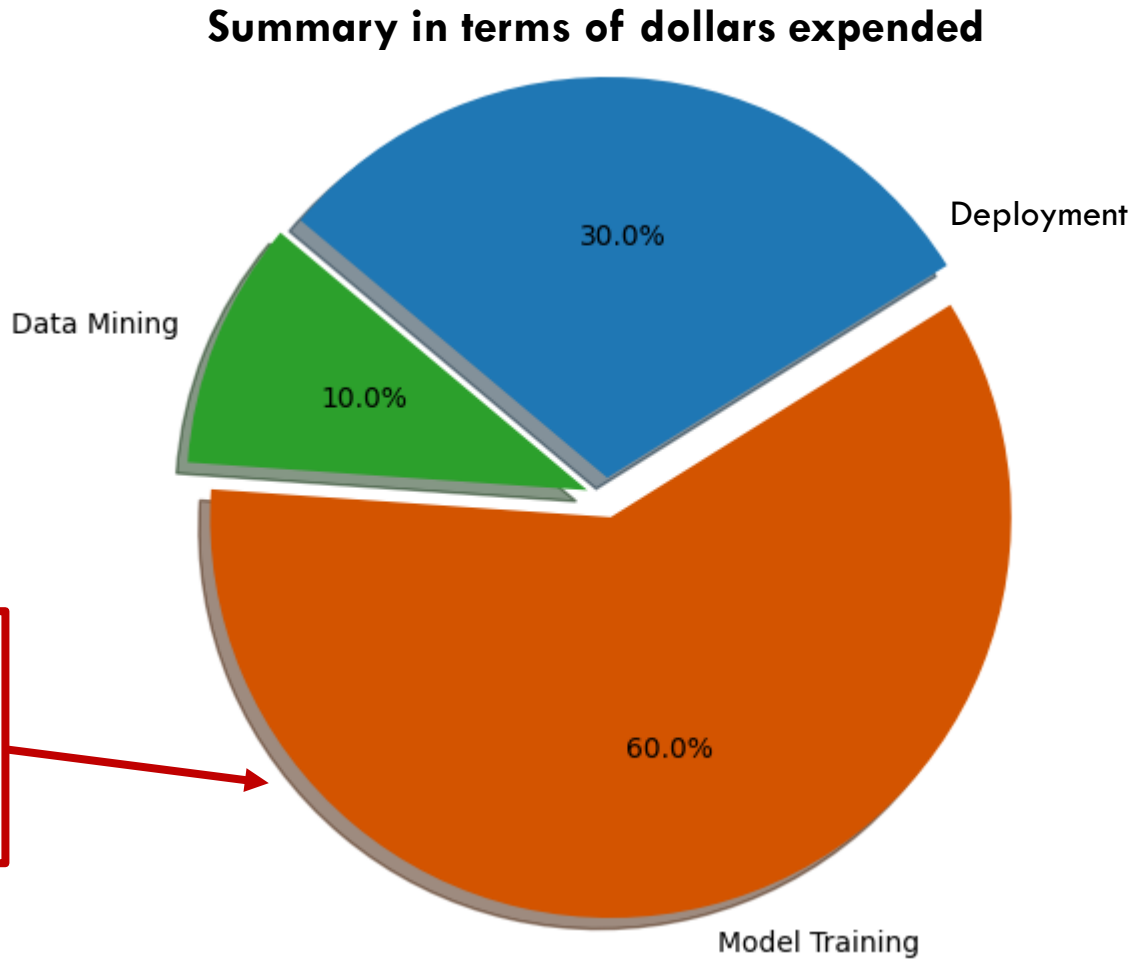
Probably requires just one cycle of training that would start from our earlier solution. **Costs $2K**.

SUMMARY: HUMAN EXPENDITURE OF TIME DURING A DEVELOPMENT AND TRAINING EFFORT



Assumes just 10 human cycles of the ML design/training process. Sometimes as the dev team revises its approach the data mining tasks are repeated too.

… AND COSTS ON A CLOUD PLATFORM

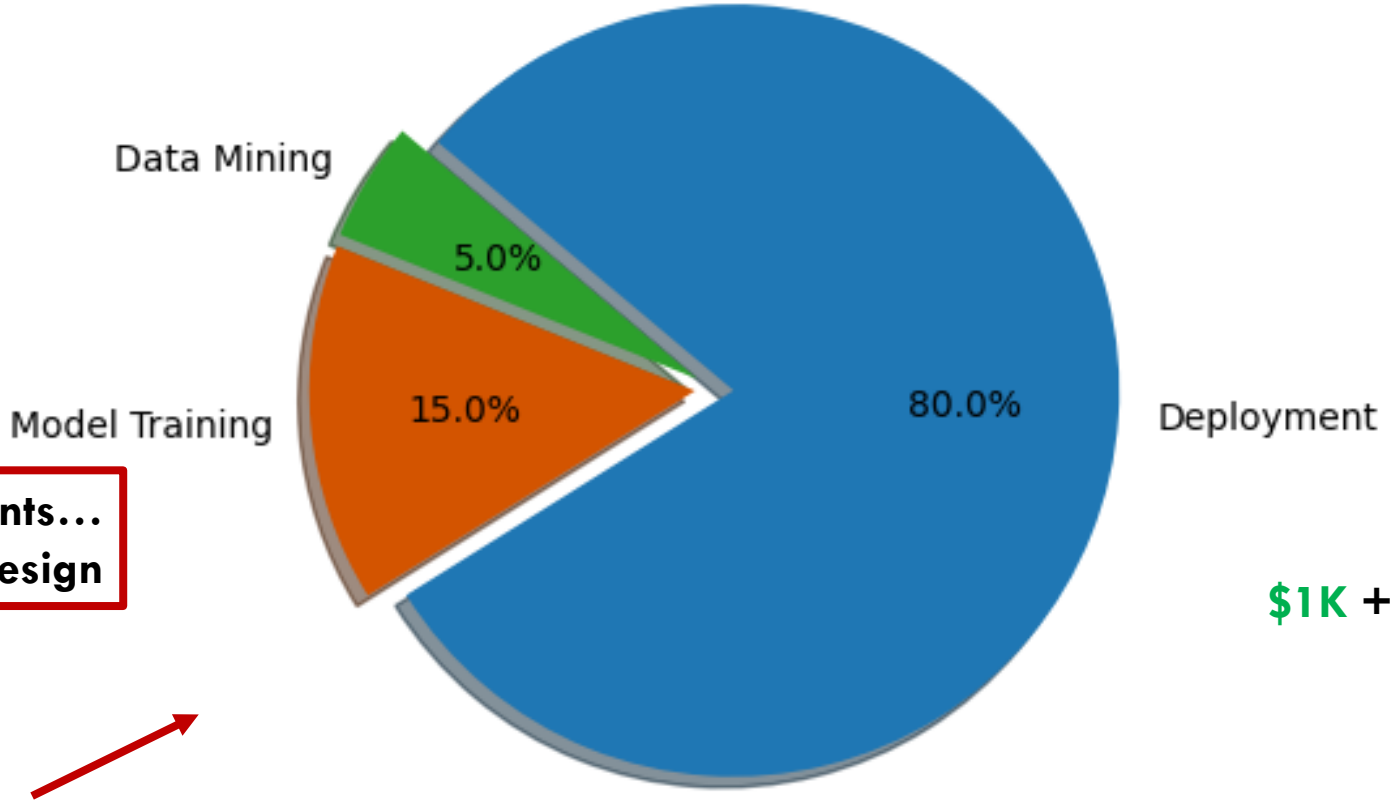


Could grow if we need more development and debugging cycles!

$1K + $100K + $50K

TWEAKING IS CHEAPER

We can reuse the MoE design and weights and only do a single data mining and training cycle.



Model adjustments… not a full redesign

$1K + $10K + $50K

SO, WHERE SHOULD WE FOCUS EFFORT IF WE WANT TO OPTIMIZE USER EXPERIENCE AND COST?

In 2025, the core question centers on supervised fine tuning and other forms of RLHF training.

- These are early days for ML and everyone is hard at work in the "many cycles of developer effort" stage
- Integrating AI into "everything" will take a decade or more
- Yet over time, model serving and tweaking becomes more dominating. Call this a 2035 concern.

LET'S LOOK MORE CLOSELY AT SFT FOR A RAG MOE LRM

When doing SFT for a LLM, for example to train a low rank adaptor (LoRA), the limiting factor centers on pure GEMM computations occurring on the GPU

This is because the data and the current model all fit in the GPU and the optimizer can iterate with no help from the host.

The pattern is quite different when we do MoE training

EXAMPLE: HYBRIDFLOW SYSTEM

HybridFlow: A Flexible and Efficient RLHF Framework

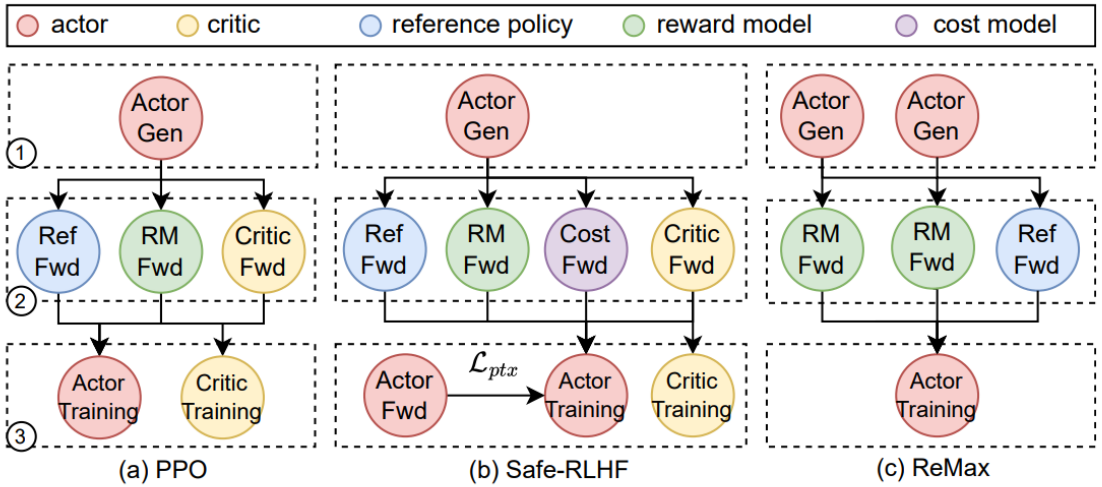


Figure 1. Dataflow graph of 3 RLHF algorithms [19, 43, 55]. Stage ①, ②, ③ represent Generation, Preparation, and Training, respectively.

Central observation: we have multiple MLs that cooperate in each iteration:

- Actor to generate next token
- Cost model uses classic LLM scoring to assess the quality of the proposed output suffix
- Reference model judges quality of the output sequence using a much larger expert
- Critic assesses progress using customer-specific criteria

HYBRIDFLOW WITH RAG MoE TARGET

Here we will have a family of models each contributing an "opinion" that contributes to the generated output

Contents of the RAG database must also shape actions: MoE must query the vector database in a way that will match the desired documents (and not undesired ones)

In effect we have the HybridFlow pattern instantiated multiple times, with a form of hyperparameter search too (to optimize expert weights)

ITERATIVE EXECUTION PATTERN

As the optimizer runs, it builds responses to queries token by token, scoring the quality in multiple ways, then adjusting the model.

This “autoregressive iterative process” dominates the SFT runtime… and uses inference repeatedly (for generation, scoring, critiquing)

Leading to the question: What can we do to speed up inference? This will pay off during SFT and while hosting the solution, too.

IN COMPLEX CASES THIS PATTERN BECOMES A DISTRIBUTED ONE, ON A COMPUTE CLUSTER

Today, the industry has extensive experience with RLFT and SFT on a monolithic ML (one that fits entirely on one host).

- The one host might have multiple GPUs, yet once the job is loaded on that host, it runs without needing data or help from other hosts
- The issue is that over time, the pattern will evolve and the workflow will be more dominated by training on host clusters.

HOW BIG WOULD THESE CLUSTERS BE?

Once we "break the walls" from being monolithic to spreading over multiple machines, we'll quickly have some fairly big jobs

Several communication patterns arise: point to point messaging, AllGather, AllReduce (not everything is shown on the figures)

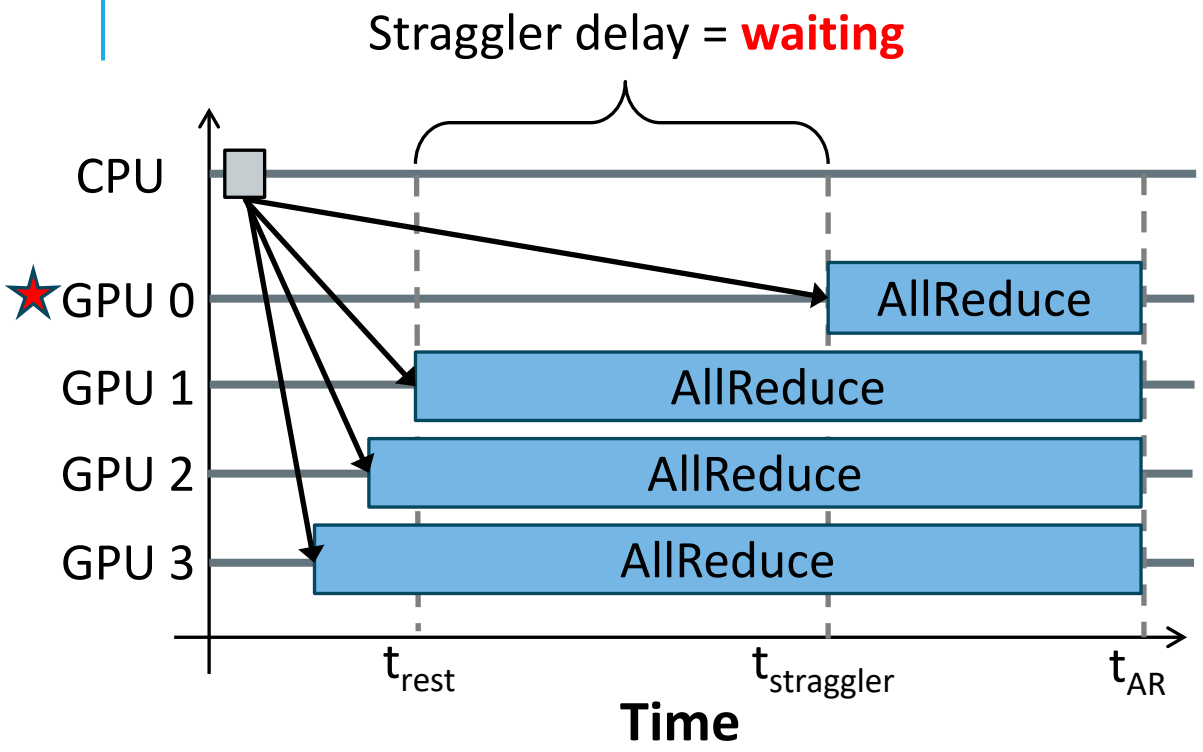
Latency of component-to-component interactions will play a increasingly dominant role in shaping performance.

… POSING A QUESTION (ONE FOR WHICH NO ANSWER IS KNOWN, AS OF 2025)

How significant will these inter-machine messaging costs be relative to the full cost of running SFT on this complex model?

Will we be almost entirely dominated by GEMM mathematics? Some people assume this is likely.

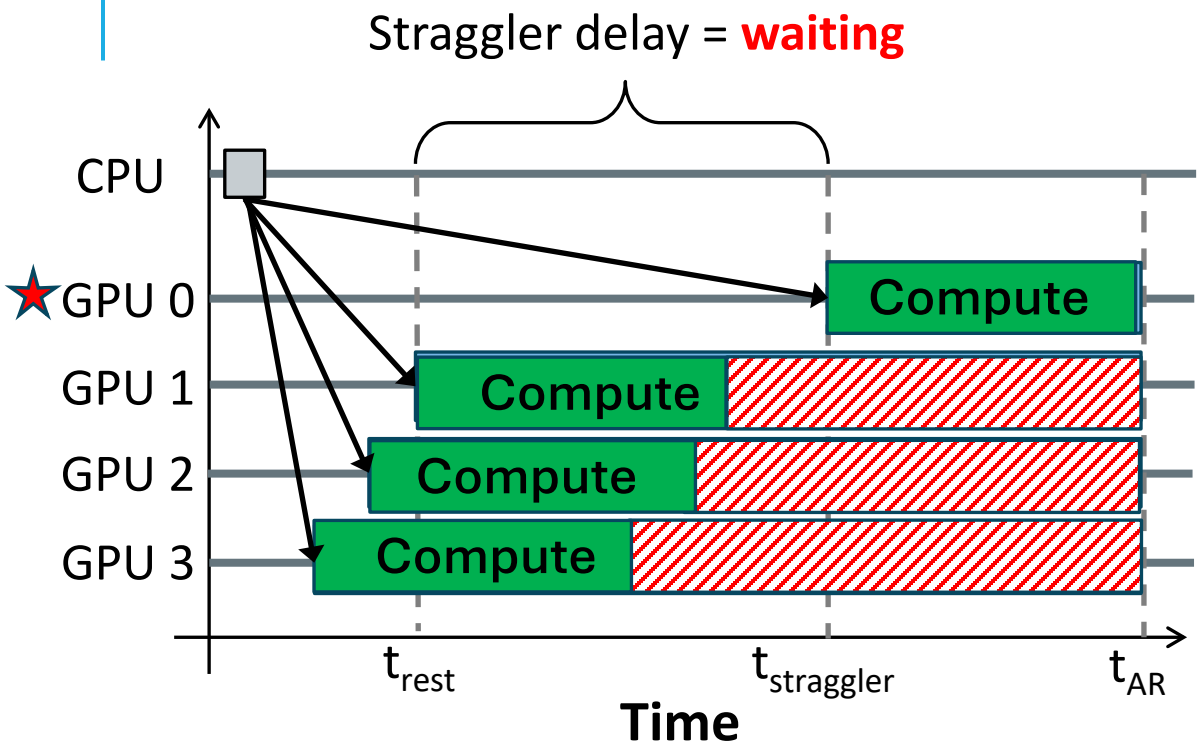
OBSERVATION BY RACHEE SINGH: STRAGGLER GPUS



In a 4 GPU server, straggler is late by 10 milliseconds

- AllReduce ends with each worker collecting data from all the others and then reducing (aggregating)
- We think of this as a synchronous computation limited in part by network speed. Rachee found that late results (here, from GPU 0) often bottleneck the full computation!

OBSERVATION BY RACHEE SINGH: STRAGGLER GPUS



In a 4 GPU server, straggler is late by 10 milliseconds

We "own" four GPUs, yet obtained 1.5 GPUs of work from them. Most profilers would claim all four were in use the whole time!

WHY IS THIS SO DIFFERENT FROM SMALL MODEL TRAINING OR SFT LORA TRAINING?

The phenomenon is less common if the host and GPUs are all a single system

With a compute cluster, we are at much higher risk of significant costs from the way that training interacts with AllReduce, Linux and the network

AT CORNELL, WE CREATED VORTEX TO EXPERIMENT ON REDUCING SUCH COSTS

Rachee's group is distinct from mine… she explores stragglers

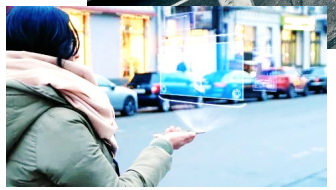
The Vortex examples that follow are from inference scenarios and RAG knowledge retrieval, not SFT for a RAG MoE.

Just the same, we think the insights will translate over

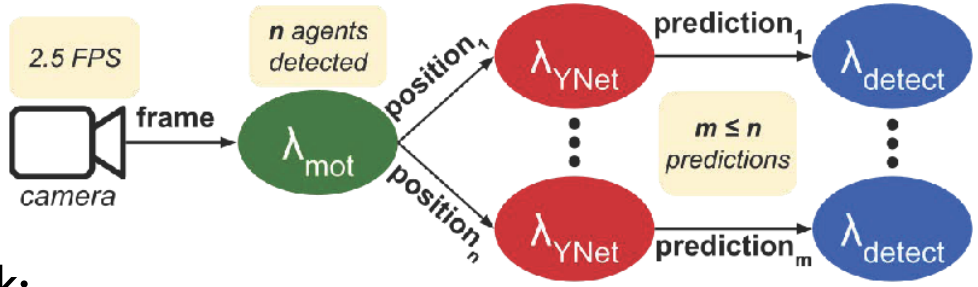
Key people: Sagar Jha, Weijia Song, Alicia Yang. But many others had significant roles.

EXAMPLE 1: URBAN TRAFFIC PLANNER WATCHING FOR ISSUES AT AN INTERSECTION

Problematic intersection



City traffic planner



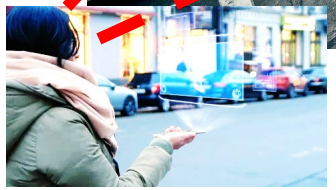
ML task:

- Segment each video frame, "agents" are entities moving under independent control
- For each agent, run a pipeline. The red ML predicts its trajectory, emitting a stream of coordinates in order of likelihood
- The blue risk detector senses dangerous proximity

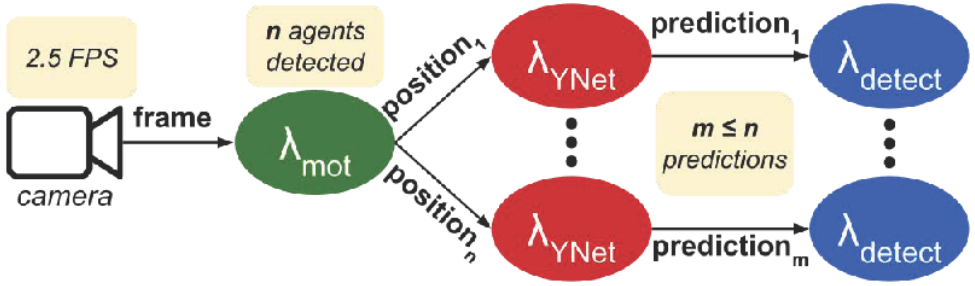
Output: an alert visible to the city traffic planner

EXAMPLE 1: URBAN TRAFFIC PLANNER WATCHING FOR ISSUES AT AN INTERSECTION

Problematic intersection



City traffic planner



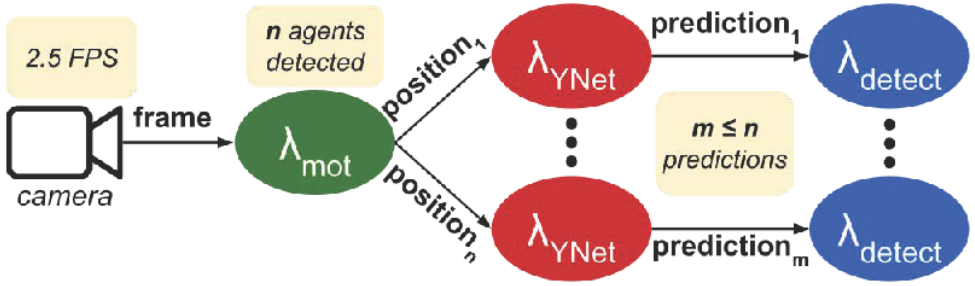
We can flash the alert on her hololens

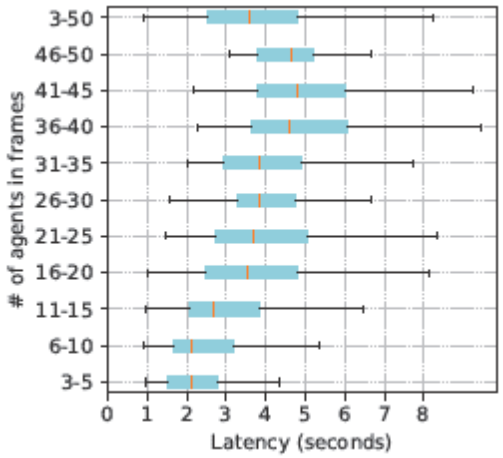
But it needs to be timely or the scene will have evolved too much for this to be useful

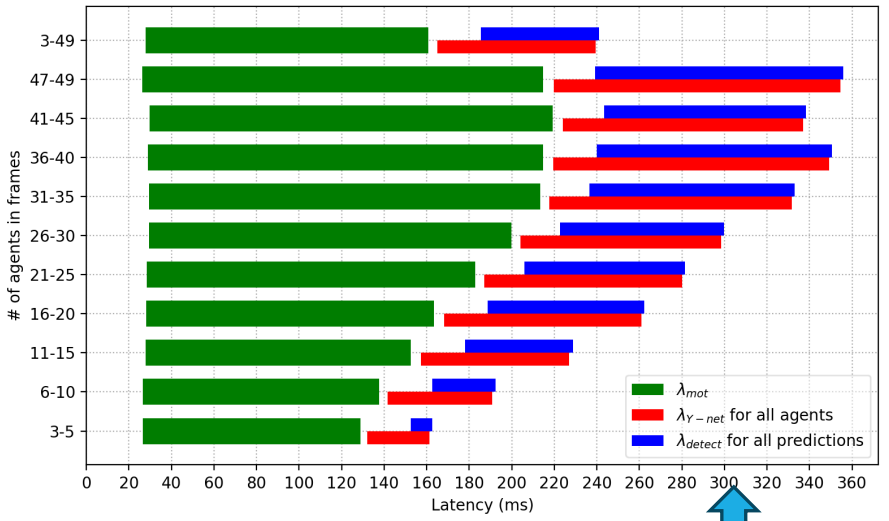
EXAMPLE 1: URBAN TRAFFIC PLANNER WATCHING FOR ISSUES AT AN INTERSECTION

Problematic intersection

City traffic planner







Azure: Seconds. Vortex: Milliseconds

NOTICE THAT WE ARE TALKING ABOUT LATENCY AS WELL AS THROUGHPUT

The urban traffic planner wants to be warned about events "in the moment", not a minute later

During RL training and SFT we traditionally focus on asynchronous training runs and the main metric is throughput

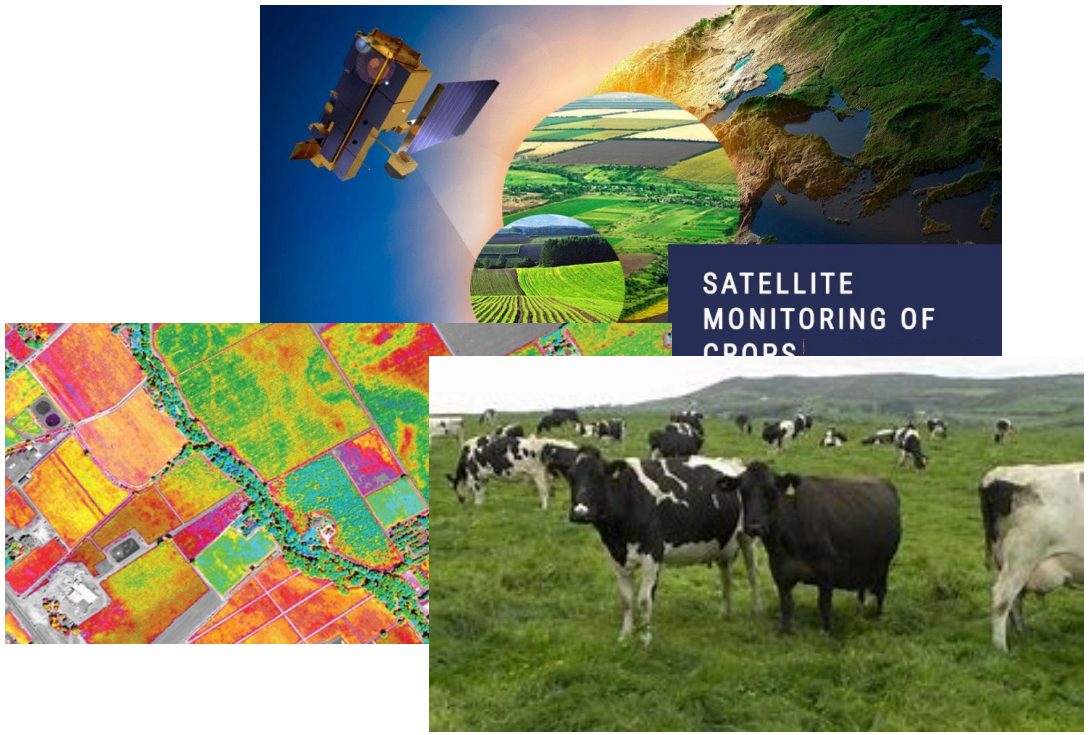
Yet latency during the autoregressive iteration will actually shape the performance of the full run

… AND THIS IS COMMON

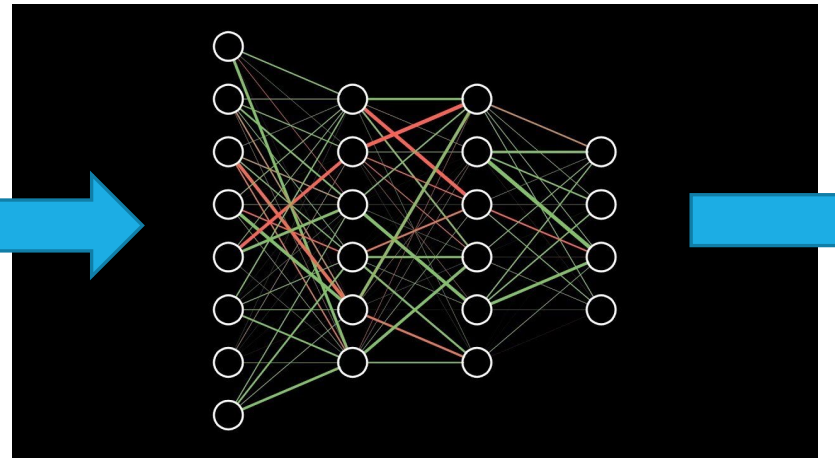
As AI is fully integrated into the real world, latency goals arise

FOR EXAMPLE, IN FARMBEATS

Data sources



Cooperative ML
Distributed AI

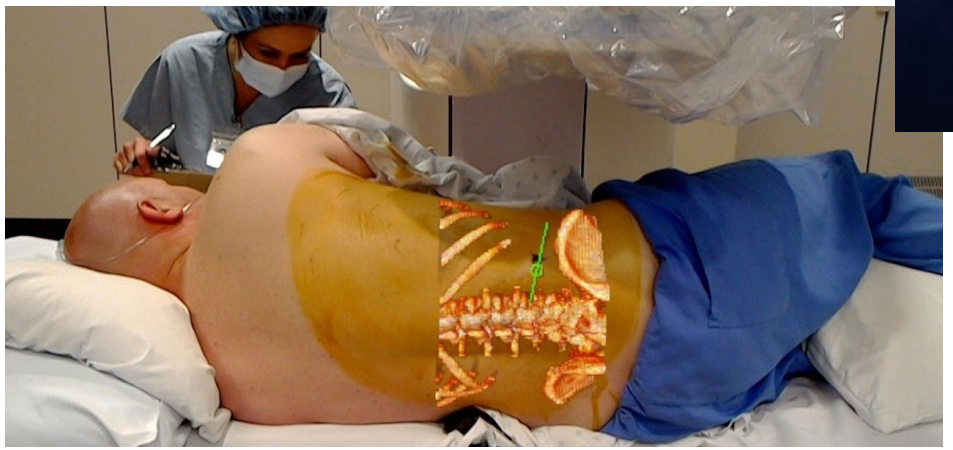


Smart
Farming

This cow may need medical attention

… SERVICING EQUIPMENT

… MEDICAL AI ASSISTANT

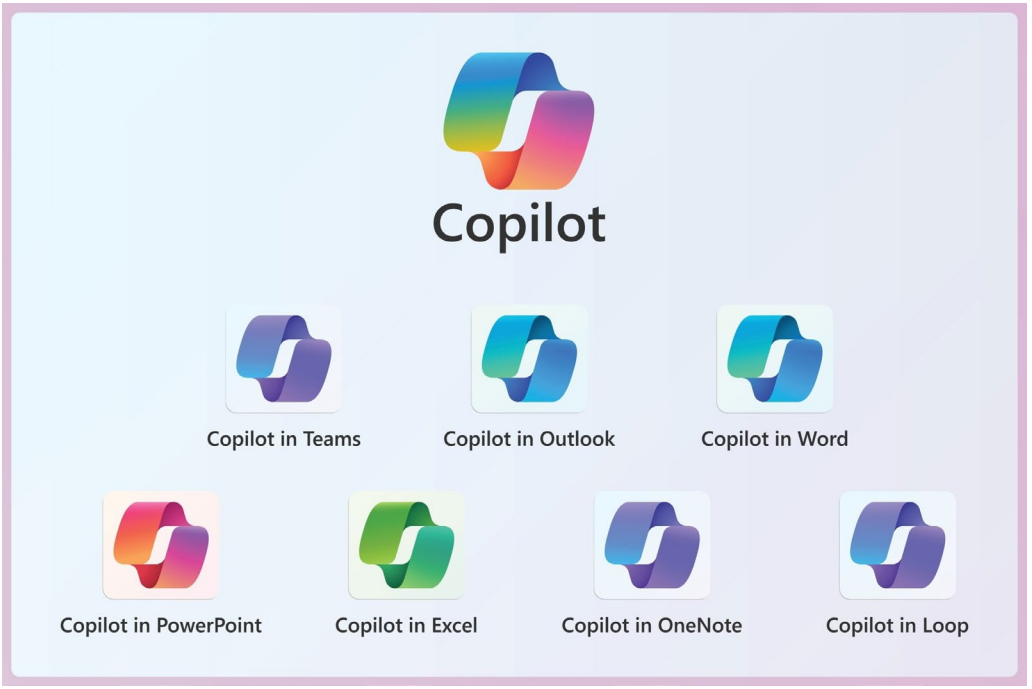


… AND EVEN IN ROUTINE COPILOT SCENARIOS!

AI copilots treat ML as a service

Query rates will be very high
(with a copilot, the AI issues them)

User experience shaped by speed



HOW DID VORTEX REDUCE LATENCY SO MUCH?

We didn't modify the AI pipeline components: they came from open source repositories (Hugging Face). We "wired them together" to create this data flow.

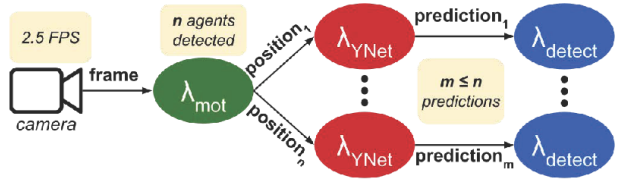
… We didn't even fine-tune them!

Instead, we focused on **platform components** the pipeline uses

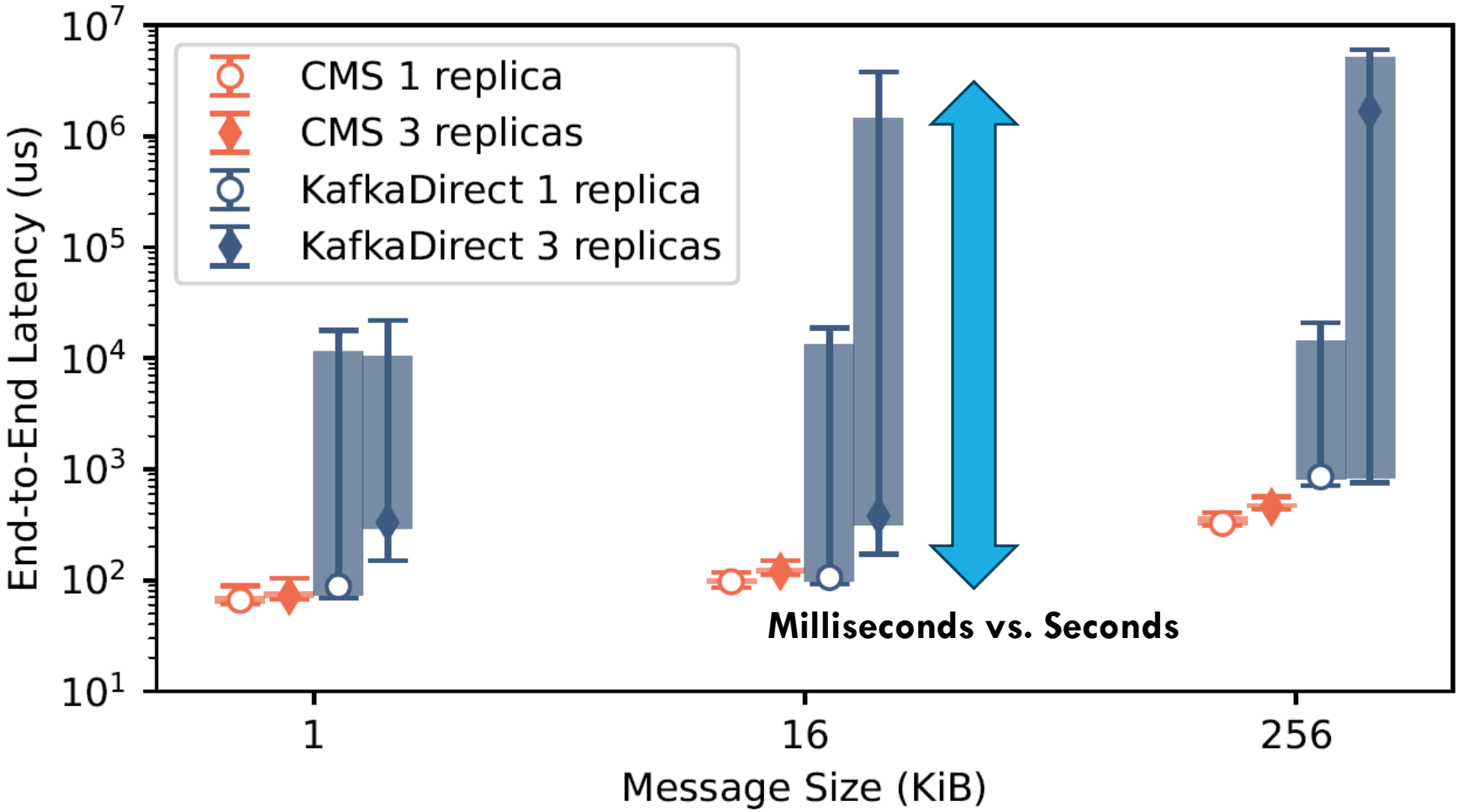
EXAMPLE: MESSAGE QUEUING DELAY

Recall from our lectures on Apache that Kafka is a widely used message queuing service, with two modes: a notification mode (DDS) and a transactional stored-message mode (persistence)

Azure has a message queueing service too. We used it to connect components of our AI pipeline. Output of one is input to the next



LAB EXPERIMENT USING KAFKA-DIRECT… CONSIDERED FASTEST AMONG ALL OPTIONS YET HAS HUGE TAILS!



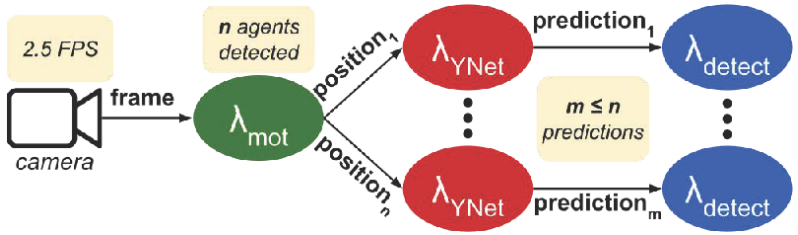
IMPLICATION?

The Vortex CMS (an earlier version of our Kafka API) is far outperforming Kafka Direct: A version of Kafka running on RDMA.

Kafka Direct is not the Azure Queuing Service… but supposedly outperforms that Azure service, when RDMA is available.

So here we are seeing that these solutions do a poor job when people build ML pipelines in the “standard way”

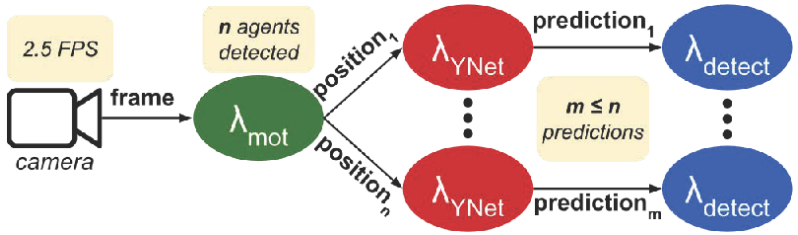
THIS PIPELINE ALSO SHOWS OBJECT/TASK "AFFINITY"



This is a program-level version of "keep your friends close"

Consider the ML pipeline used for traffic hazard sensing. Which ML tasks will need which objects?

THIS PIPELINE ALSO SHOWS OBJECT/TASK "AFFINITY"

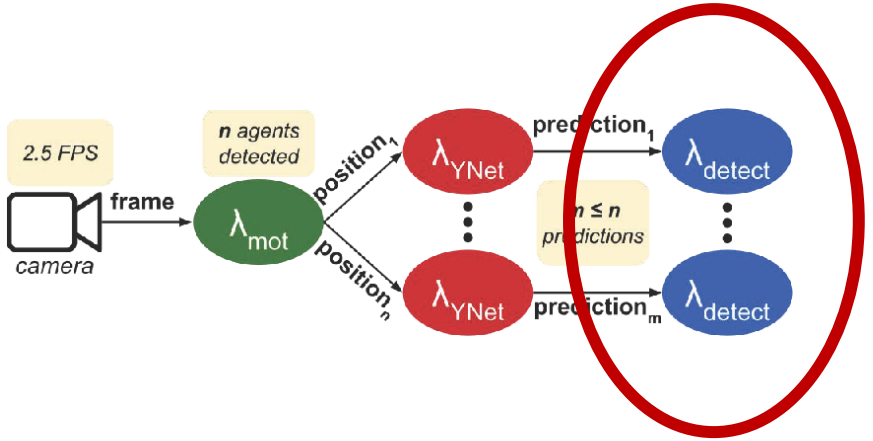


First, understand that this ML pipeline spawns one red and one blue AI task per "agent" in the image (vehicle, pedestrian, …)

Roles: Green "segments" the video, red predicts trajectories for each moving object, and blue looks for potential collisions

So with 30-50 moving objects it has > 100 VML tasks.

THIS PIPELINE ALSO SHOWS OBJECT/TASK "AFFINITY"



Location predictions for each agent will be consumed by collision detectors for agents "nearby"

Vortex offers a way to group objects that are likely to be accessed all at once. We call these "affinity sets"

Affinity grouping requires some replication but led to big speedups in the collision detection stage

IMPROVING LATENCY IN THESE WAYS ALSO IMPROVED THROUGHPUT!

Often people assume that we should focus on throughput and ignore latency, but those studies were on older LRM training systems.

Training and running these more complex ML pipelines seems to require a new way of thinking about performance

COSTS OF PIPELINED MLS

Modern MLs are almost always pipelines of components that communicate over layers like Kafka

Inefficiencies arise if we are inattentive to the way these behave. But with attention, pipelines outperform monolithic MLs.

Seemingly minor platform features can cost more than the ML!

FRENCH AIR TRAFFIC CONTROL SYSTEM

Air traffic controllers update flight plans

Flight plan manager tracks current and past flight plan versions. Replicated for ultra-high reliability.

Message bus

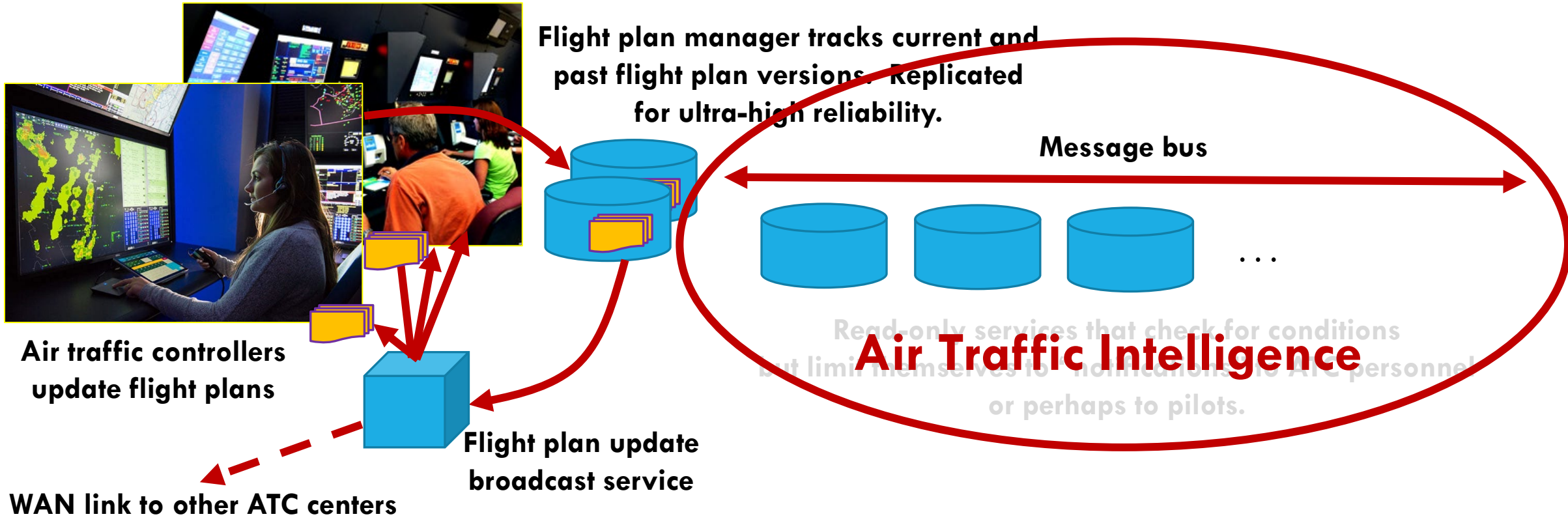
. . .

Read-only services that check for conditions but limit themselves to "notifications" to ATC personnel or perhaps to pilots.

Flight plan update broadcast service

WAN link to other ATC centers

FRENCH AIR TRAFFIC CONTROL SYSTEM



WHAT PLATFORM ISSUE ARISES HERE?

This system used a faster pub-sub product, so the problem we saw with Kafka wasn't a problem

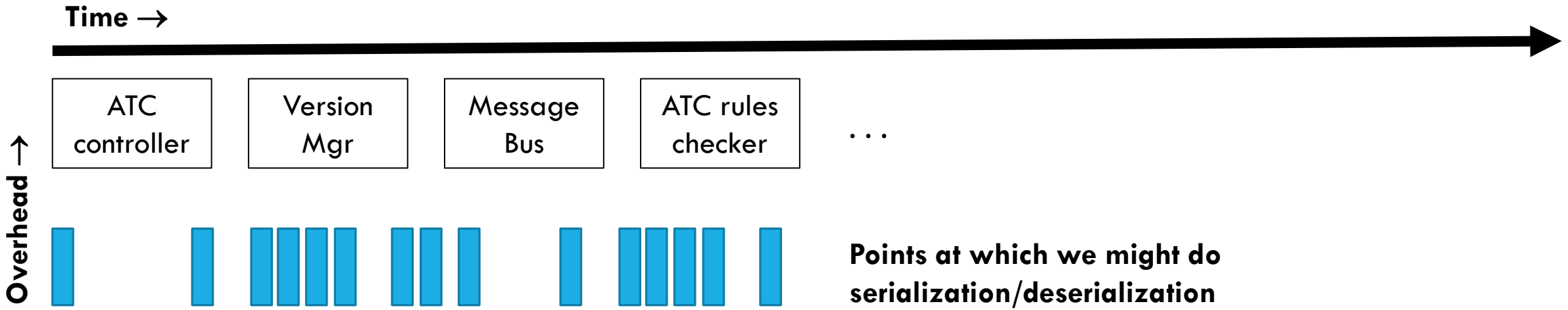
But it has a great many cross-module and cross-language events. Air traffic control leverages code in old languages (Fortran, Ada) as well as the usual modern ones (C++, Java, Python)

All support cross-module sharing, but…

SERIALIZATION LIMITS PERFORMANCE!

Each time an object is read or written (from disk or network)

Each time an object is passed from one module to another



A FLIGHT PLAN IS ESPECIALLY COSTLY!

A single flight plan object can be 10-15MB, 1000's of fields.

Why so many? Economic issue: Each flight plan incurs a big fee for initial filing. Airlines try to reuse a single plan even across totally unrelated flights!

en reserializes to pass to

ans are read only!

To change the flight plan, the current controller owning that plan interacts with the version "database" and the old version is then replaced, system-wide. No other component can update a flight plan.

BUT WE CAN HIDE THIS COST…

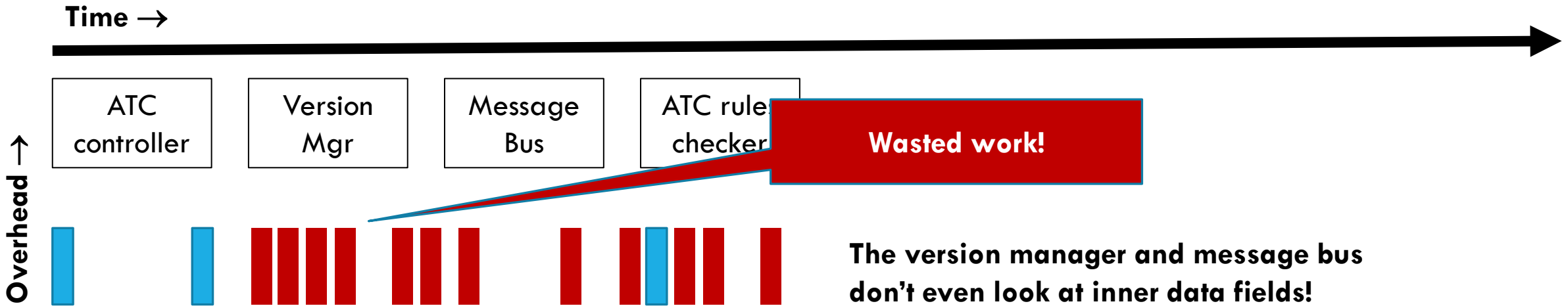
Suppose we modify the flight plan into an object loaded lazily when a field is actually read?

Now we have a compact and an expanded representation

ALL THE RED EVENTS WERE UNNEEDED!

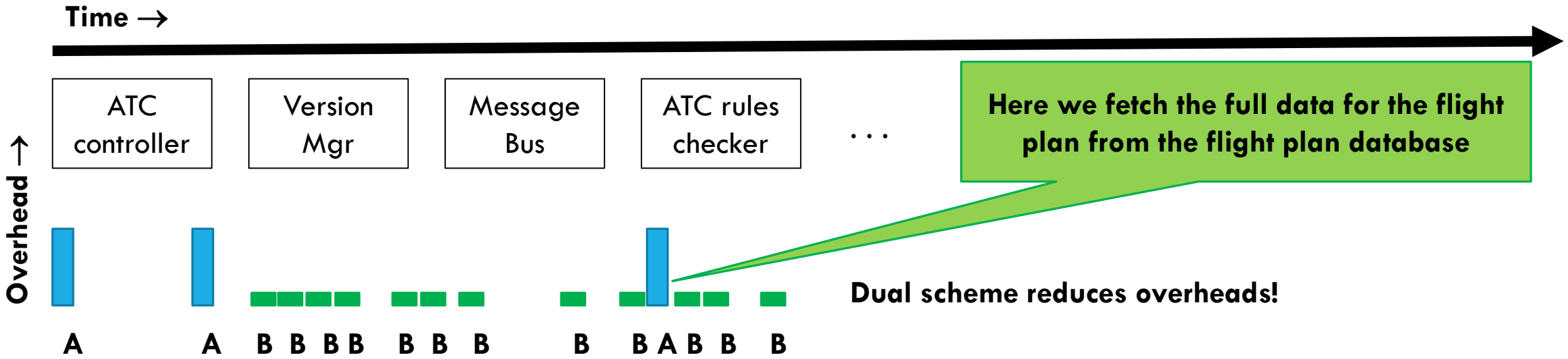
Each time an object is read or written (from disk or network)

Each time an object is passed from one module to another



SAME SOLUTION WITH A LAZY GETTER

We didn't discard serialization or deserialization. *But it happens much less often.* And the objects moved from component to component are tiny URLs, not 15MB flight plans.



THIS WAS A REAL EXAMPLE!

It arose in a upgrade of the French ATC system, called 4-Flight, slated for full rollout in 2026

The entire project was bottlenecked on this serialization cost, but the team perceived it as slowness in the new AI layer

Nobody realized that platform overheads dominated all else!

DON'T AI ALGORITHMS MATTER MOST?

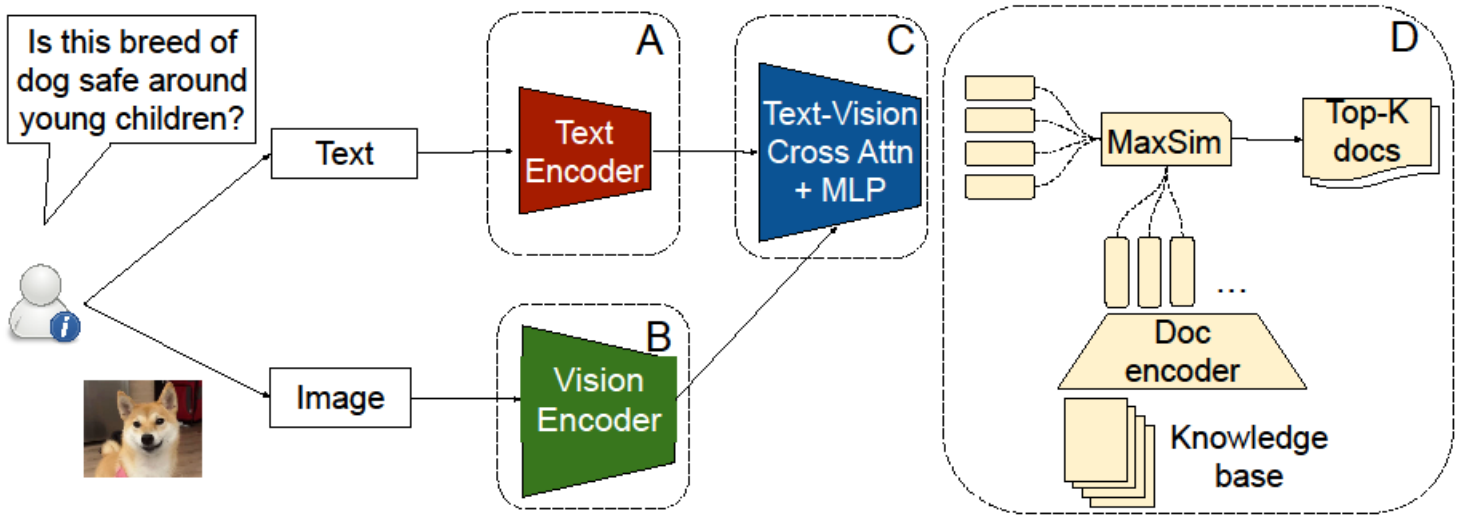
Algorithms *do* matter, a lot. And their speed is crucial

But we are seeing how implementation choices become more dominant when an ML is running as a pipeline on multiple hosts.

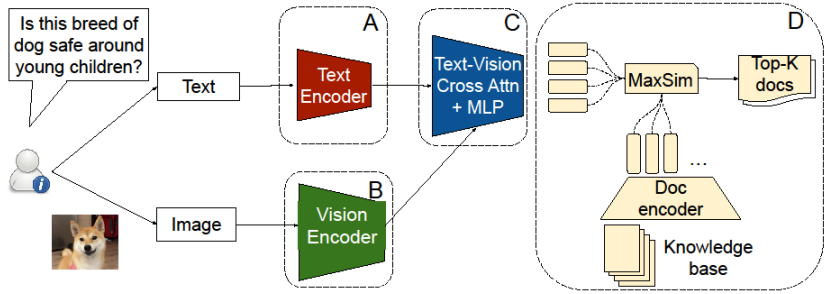
SFT that fits into a single host with its 4 or 8 GPUs has distracted us from the future challenge of SFT for RAG MoE LRMs

CONSIDER PREFMLR

This is an ML as a Service constructed from a popular RAG LLM pipeline called FMLR by removing its generative response step



PREFMLR IS EASY TO EXPRESS IN PYTORCH



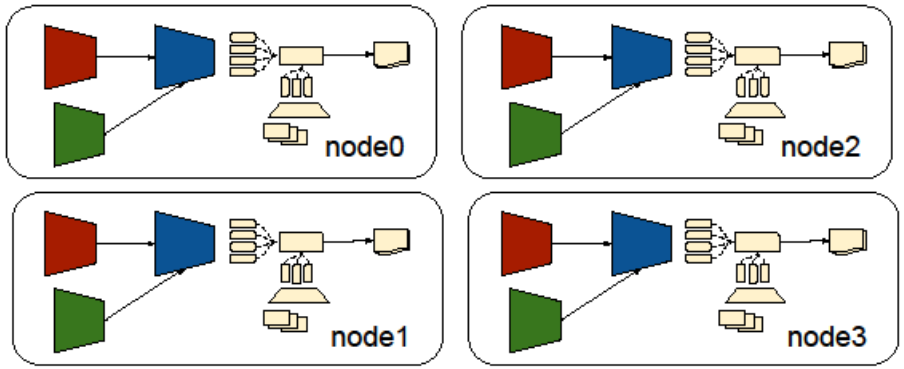
The technology is very componentized: PreFMLR computes D(C(A(Q), B(Q)) for preexisting, standard subtasks.

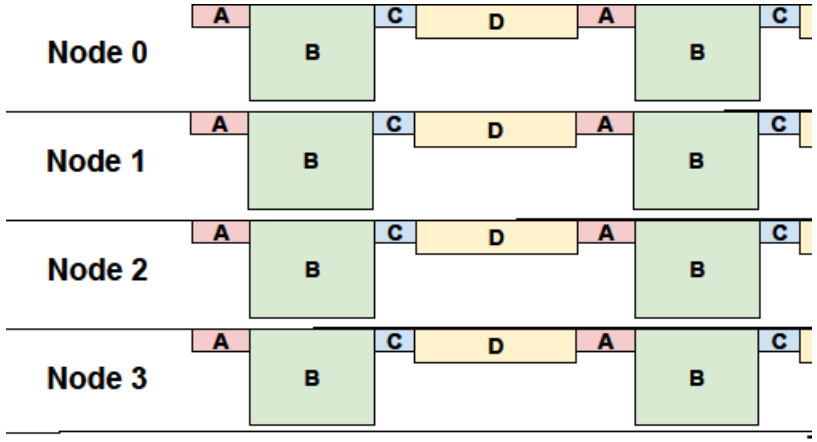
The high level ML looks very elegant and clean, yet a lot is going on under the hood.

High level coding is easier, yet often trusts that the runtime framework will somehow "guess" the optimal runtime strategy

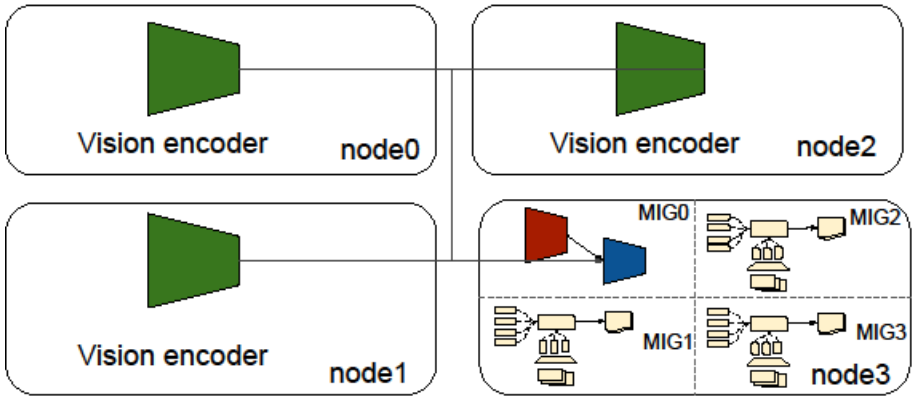
… BUT HOW WELL DOES THAT RUNTIME DO?

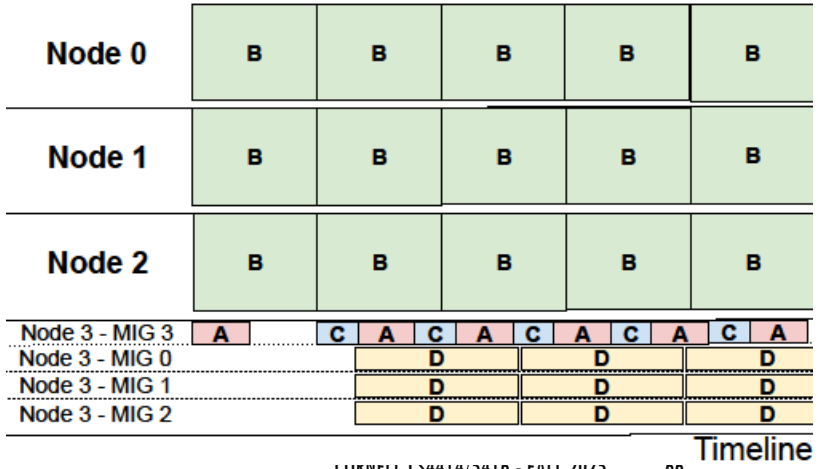
Monolithic: every node runs all four components





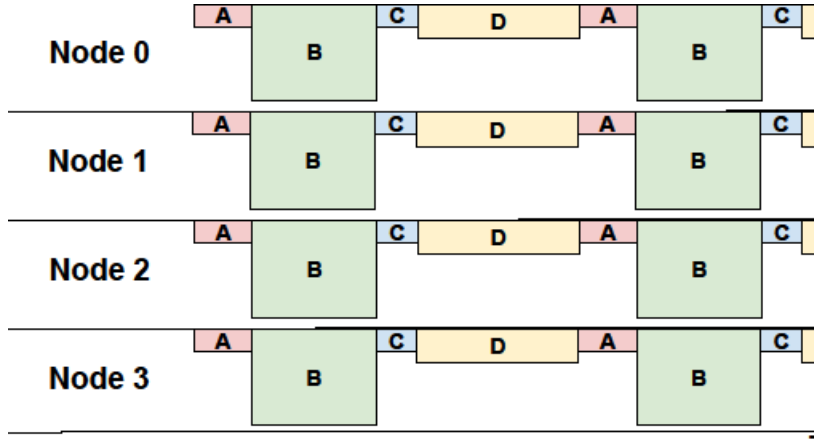
Microservices: Nodes 0,1,2 only run B Node 3 runs A, C and D



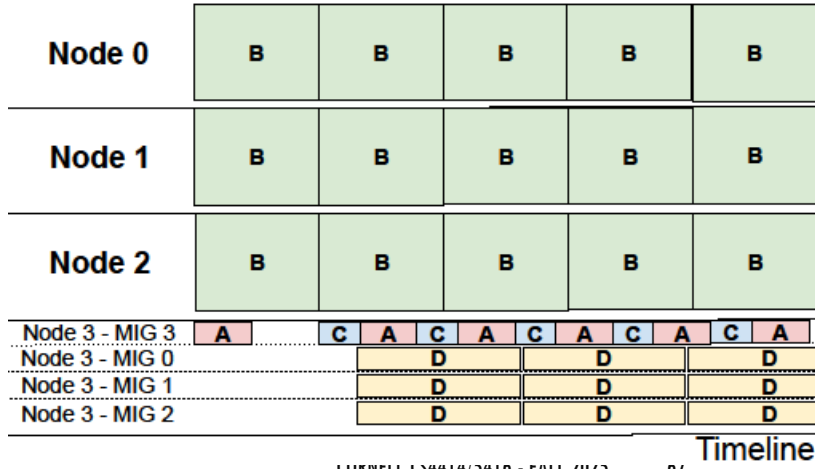


… BUT HOW WELL DOES THAT RUNTIME DO?

The Microservice deployment handles 15 queries in the time the monolithic one needed for 8



Monolithic deployments are widely prevalent today, and yet make choices that greatly inflate costs and harm overall performance



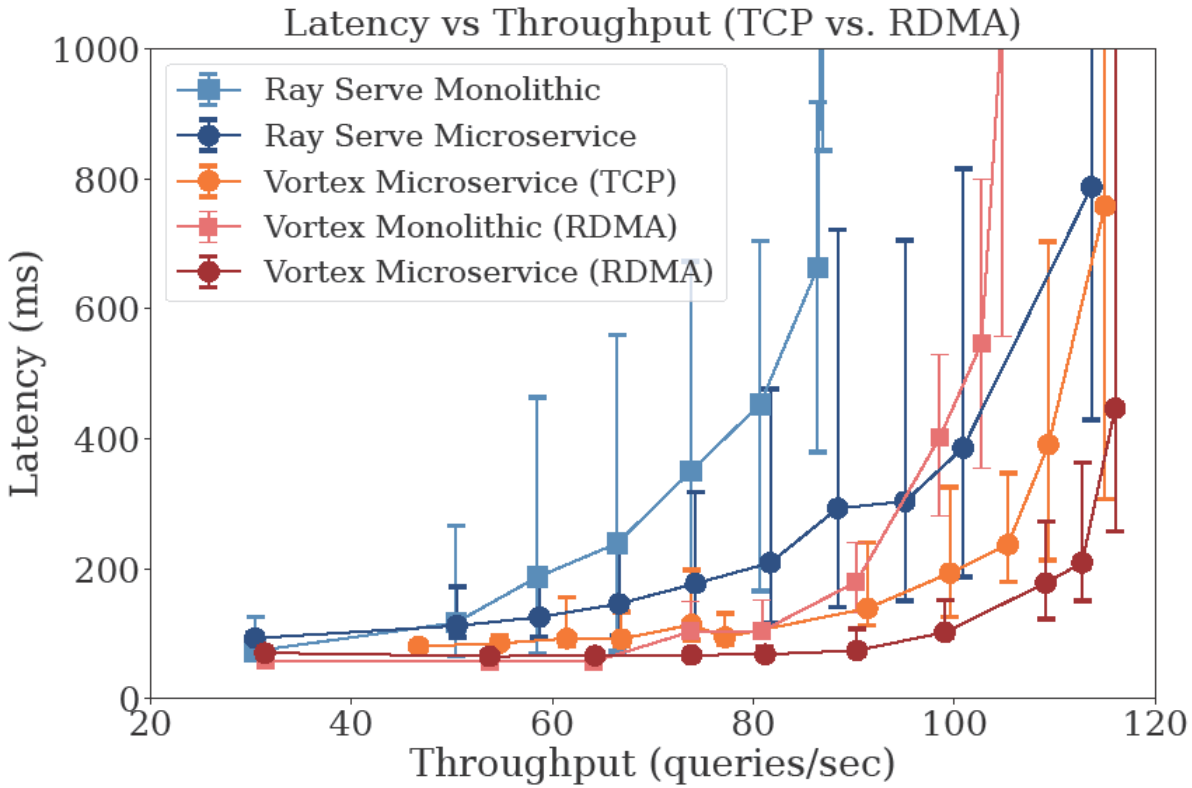
DO OPTIMIZATIONS LIKE THIS HARM LATENCY?

Recall that the bottom line is the time to do a cycle of RL training or SFT, or to respond to a query in a deployed system

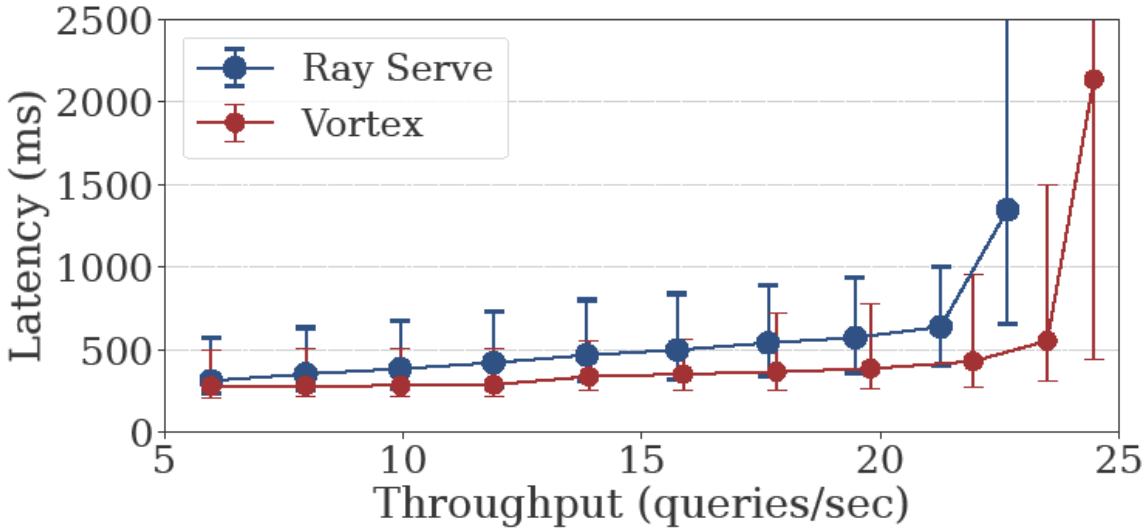
Are we harming latency when we improve hardware efficiency by concurrently sharing GPUs between different tasks?

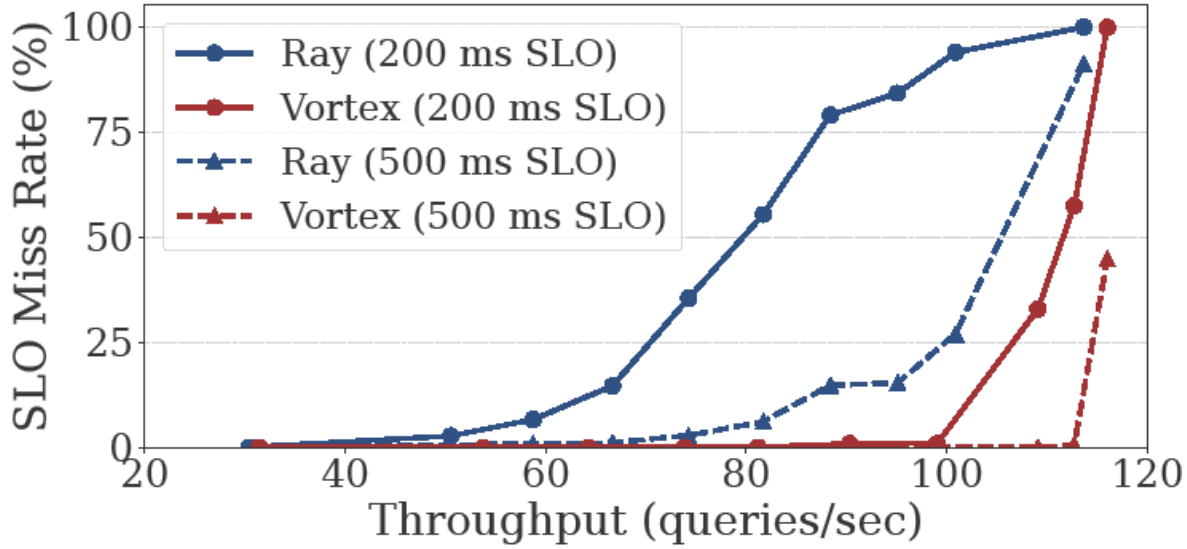
A SERVICE LEVEL OBJECTIVE (SLO) IS THE LATENCY ACHIEVABLE AT A GIVEN RATE

Here we can see that Vortex also responds quickly on queries: lower latency, tighter error bars. The RDMA option is best…



GIVEN SUCH A GRAPH, AN ML SERVICE CAN OFFER AN SLO (LATENCY, AND A LIMIT ON MISS RATE)





We can achieve **higher throughput** and yet offer **lower latencies** – better "SLOs", all while using the ML hardware more cost-effectively!

SOME OF THESE OPTIMIZATIONS WERE HARD!

Quite a few of the steps required here were actually easy.

But they become more challenging when they involve transparently leveraging the hardware more effectively, and at that level can turn into genuine systems research topics

OPTIMIZATIONS WE EXPLORED

Asynchronous data flow with as few locks as possible

Cache intermediary results close to MLs that will need them.

Awareness of the huge slowdown effects of non-local memory accesses on NUMA processors

Batching opportunistically, not using a static batching pattern

Leveraging RDMA

Using just one thread per concurrent sub-task. Avoids sharing data between those tasks to reduce/eliminate locks

Avoiding copying (memcpy, arguments by value, etc. can be a huge overhead)

Replication with strong consistency

Placing objects and tasks so the needed objects will be locally available

Native APIs focus on a key-value store but has wrappers for Kafka, POSIX

OPTIMIZATIONS WE EXPLORED

easy

Which are of these are easy to do – perhaps tedious or annoying, but conceptually straightforward?

medium

Which require more thought?

Major undertaking

Which require significant full stack redesigns – perhaps hidden from the ML and with no change in the PyTorch or Tensor Flow APIs, but still a lot of effort?

EXAMPLES OF OPTIMIZATIONS THAT ARE NEEDED IN CLOUD PLATFORMS

Asynchronous data flow with as few locks as possible

Cache intermediary results close to MLs that will need them.

Awareness of the huge slowdown effects of non-local memory accesses on NUMA processors

Batching opportunistically, not using a static batching pattern

Leveraging RDMA

Using just one thread per concurrent sub-task. Avoids sharing data between those tasks to reduce/eliminate locks

Avoiding copying (memcpy, arguments by value, etc. can be a huge overhead)

Replication with strong consistency

Placing objects and tasks so the needed objects will be locally available

Native APIs focus on a key-value store but has wrappers for Kafka, POSIX

BOTTOM LINE?

Evolution of ML is changing the game…

Bottlenecks inflate the cost of training and harm the customer experience long after the solution is deployed.

This argues for building **expertise in systems for ML**. The work isn't always research but some steps are hard. The investment pays off through cost efficiencies and speedups