Rethinking Observability for AI Workloads on Multi-tenant Public Clouds

Theophilus A. Benson
Carnegie Mellon University

ABSTRACT

Today, a majority of AI workloads (LLM and D) are trained and used on public clouds where the ML-engineers, i.e, tenants, naturally, do not have sufficient information and control over the cloud infrastructure. Yet, cloud infrastructure level errors, while small, have the largest impact and can result in significant monetary loss ($1.8-30K) and wasted time. Most diagnosis efforts are manual which further exacerbates the situations and the few attempts at automated assume full control over the entire ecosystem – not true in public clouds.

In this vision paper, we briefly survey recent reliability publications from 4 hyperscalers which allows us to highlight key challenges regarding debugging workloads in general and specifically in a multi-tenant public cloud environments. With these insights, we sketch a solution for diagnosis and explore the design space of cooperative approaches to provide sufficient information for this solution.

1 INTRODUCTION

Significantly large infrastructures are devoting to training and inferences jobs, with recent data centers containing over 100,000 Nvidia H100 GPUs. At these scales, job failures and inefficiencies are costly, with recent sources claiming between 1700-20,000$ per hour for customers for large clouds [7, 17]. However, today, diagnosis is mostly manual [9, 12] taking 1-3 days **(Table 1)** and failure is addressed by analyzing data rolling back to the last checkpoint and restarting the job . The efforts are wasteful for several reasons: first, during the days required to resolve the failure, GPUs are idle; second, work between the check pointing and failure is wasted and needs to be re-run; third, re-running a job evicts lower priority jobs.

Despite the significant attention that LLM and learning job failures have received, there are few automated solutions for detecting failures [12, 20]. Moreover, none of these solutions explicitly address the unique privacy and data-sharing concerns for the cloud. The most related work, e.g., SuperBench [20] and ACME [12], assumes full access to tenant workloads and leverages analysis to proactively detect faults. However, for many clouds, the tenant and the provider do not freely share all the information. Additionally, as both works admit, detection can be limited due to the rapidly evolving ecosystem.

In this work, we lay down the vision for an ensemble of methods to reduce to costs of failures by combining proactive technique to detect failures before a job runs and reactive techniques to detect impending failures, thus reducing wasted resources. Our work builds on several key characteristics of emerging LLM-workloads:

	Bytedance [7]	Bing [20]
1-days	36%	61.9%
3-day	54%	13.7%
7-day	–	10.3%
never	25%	10.3%

Table 1: Incident resolution time in several production deployments (e.g., Alibaba [17], Bings [20]).

- Rapid Evolution: The software libraries, e.g., Pytorch, evolve rapidly (i.e. 1 - 2 months), and the hardware similarly evolves rapidly (i.e., 1-2 years). Consequently, there are many version-specific bugs and often incompatibility between newer libraries and older hardware. Thus, diagnostic efforts must be made with contextual information about versions and possible configurations in mind.
- Standard Frameworks: Although rapid evolution implies a large echo system, fortunately, today training is done with a limited set of libraries (e.g., CUDNN) and tailored network stacks (e.g., CUDA). Consequently, most workloads rely on the limited set of communication and computation operators provided by the libraries and stack to efficiently realize sufficient parallelism to scale.
- Infrastructure Issues: Infrastructure class failures are an infrequence but have a magnified impact. Due to redundancies and allocation patterns, these issues often take time to manifest. Thus, we can detect their accumulation patterns before catastrophic failure.

Our vision builds on the above observation in several ways (illustrated in Figure 1): first, we leverage standardization as a means to tackle the rapid evolution. Specifically, we identify key combinations of computation and communication operators – we call these “archetypes”. Then, we identify the software version that leads to changes in this “archetypes” to further subdivide into “Sub-Archetypes”. Given these “Sub-Archetypes”, we analyze historical data to create baselines to detect the build-up patterns that lead to failures; and to detect incompatibilities between ‘Sub-Archetypes” and specific hardware versions. To address the unique challenges of

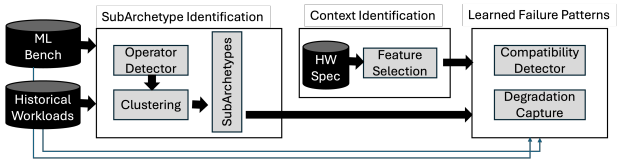

Figure 1: Pharoah's offline analysis pipeline.

public cloud scenarios, our system introduces an interface for information exchange – in particular, rather than sharing all the details of their workloads, we ask that tenants share and register Archetype signatures along with their workloads when they start job (Figure 1). Given these signatures, the cloud provider can perform analysis for tenants and provide hints about potential outages. In this work, we consider remediation out of scope; however, we believe that much of these concepts can be re-used to learn and generalize remediation.

The key contributions are as follows.

- We empirically motivate resilience challenges and highlight patterns that complicate diagnosis in the public cloud setting, where providers and tenants hold different potions of the puzzle.
- We discuss and evaluate the design space for diagnostic solutions for public clouds.
- We lay out a vision for a cooperative approach that aligns with existing cooperative models for diagnosis in the public cloud space.

2 BACKGROUND: PROBLEM CONTEXT

2.1 ML Training Pipelines

Our work builds on the insight that while AI workloads and use cases vary wildly, the data scientist and ML-engineers use a common ML tool chains of libraries and frameworks (i.e., pytorch, CUDA, ONNX, CudaNN), illustrated in Figure 2. This standardization to a common tool chain implies that the translation from user code to CUDA programs/kernel is generally done via a small set of optimizations. In particular, to improve efficiency, the various libraries and their heuristics translate ML-training into a set of fixed specific structures (tensor/matrix) and employ different permutations of a set of well-understood computation operations (e.g., multiplication method) and communication operators (e.g., all-reduce). Similarly, while hardware and their capabilities are fixed, interleaving and multiplexing between workloads is not.

Regardless, we view the use of a common tool chain and fixed hardware provide an opportunity for cost amortization – a special purpose framework encapsulating domain knowledge geared towards these common parts can be shared across users.

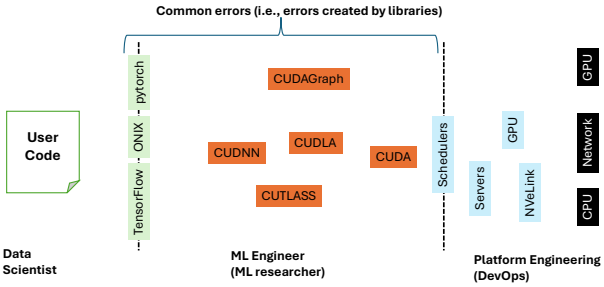

Figure 2: This figure highlights the various persona and the frameworks, libraries, and systems that they interact with.

2.2 Personas and Roles in AI Training

Abstractly, there are three roles (from a human resources perspective) involved in LLM training (we annotated Figure 2 with the role of each persona):

At the high end, data scientists write user code that selects the data, determines the training approach, and performs them on the data. These data scientists monitor their applications and evaluate its performance using loss rates.

The ML-engineers work with the ML-Pipeline of frameworks/libraries to determine which optimizations and libraries to support data scientists – these include matrix multiplication and graph management. The output of the ML-pipeline is a job that is launched and run on a cluster.

Platform engineers / DevOps manage physical resources and determine the resource allocation mechanisms used within the cluster.

2.3 Deployment Models: Public Versus Private Cloud

In this work, we explicitly focus on two models. In house, where-in all teams belong to one organization (e.g., Meta), and so there is potential more easily facilitate information exchange without legal constraints. The second, a cloud-based scenario, in this deployment setting the ML-engineers and ML-optimizers belong to a different organization than the infrastructure owner and DevOps. Consequently, the exchange of information is not supported by default and requires clear interfaces that provide very minimal information exchange. Moreover, due to legal and security concerns, the type of information is often constraint.

The above scenarios require the capture of infrastructure-level information to address training issues. However, this requires using information from the DevOps-persona to inform the ML-engineer-persona in improving their job. For cloud deployment, the question of what to export is quite a challenge. For example, with hardware error, acknowledging hardware defects can introduce legal liability. However,

	Alibaba [17]	Bing [20]	Meta [9]	Bytedance [7]
GPU	36.4%	69.5%	58%	48.6%
N/W	25%	11.1%	11.5%	14.9%

Table 2: Survey of Errors recently reported in Meta [9], Alibaba [17], Bings [20], and Bytedance [7] deployments.

resource limitations may arise because of multiplexing of physical hardware between different tenants.

3 EMPIRICAL CHARACTERISTICS

Next, we summarize insights from experience papers on production deployments, industry blogs, and conversations with ML-engineers/platform engineers.

3.1 Faults at Scale

Existing deployments on Meta, Alibaba and Azure / Bing highlight several key availability challenges (Table 2).

First, hardware errors dominate across various deployments. In particular, they fall into three classes: persistent errors – a specific device always generates errors and should be quarantined; gray failures – a device generates errors only when certainly accelerators are used and should be avoided for certain types of jobs; a transient failure – a device causes a failure due to a culmination of operational factors – simply rerunning the job should suffice. Disambiguating this difference class of errors requires historical data and statistical analysis.

Second, network errors, i.e., "NCCL Timeout", are also quite dominant. However these errors conflate a large class of root causes. For example, a memory issue that causes the kernel to kill a worker will generate an "NCCL Timeout" as with network congestion that leads to significant packet loss. Resolving these errors requires (1) identifying all workers participating in the job, (2) capturing their resources use and capturing their logs, (3) determining which worker and which resource was the bottleneck – then resizing the worker.

Third, many errors are ambiguous and could be due to decisions or tools operating in different personas. For example, "NCCL timeouts are one of the most common symptoms of a failure whose root cause could be network infrastructure, buggy model code, or other stuck components." [14] – given that this could be due to any of these, information from each persona is required to effectively disambiguate this error.

Fourth, as many have observed, certain infrastructure-level failures often have disproportional impact on the cluster. In particular, links and switch failures account for less than 2% but have a large magnified impact.

- "we see that such failures affect 0.2% of jobs. Nevertheless, we see that 18.7% of runtime is impacted by these failures" –Meta's LLAMA [9]
- " 0.057% of NIC-ToR links fail each month, and about 0.051% of ToR switches encounter critical errors and crashes. Under this high failure rate, a single LLM training job would encounter 1-2 crashes each month."– Alibaba [17]

Takeaway: Although hardware errors and infrastructure failures have a significant impact, within a public cloud setting, these signals are often obscured and hidden from the tenant and the manifest in ambiguous errors, e.g., "NCCL timeout".

3.2 Management Lessons

Given that the dominant approach is to restart and rerun jobs ideally from checkpoints. There is a significant incentive to understand and sort out why a job failed so that future failures can be avoided or delayed. Next, we summarize some of the challenges, specific to a multi-tenant cloud settings, to identify and determine the appropriate changes to ensure forward progress when the jobs are rerun.

Separation of Concerns and information Importance of Appropriate Metrics: The various personas have metrics to detect progress, e.g., ML-engineering examines the change in loss, whereas DevOps examines resource utilization. We note that often one class of metrics is insufficient. For example, a poorly coded ML-application with an error in the loop which uses just the first dataset instead of the other datasets will continue to training (thus consume resources) but will show no improvements in loss. This highlights the need for more coordination and information exchange for detection.

Importance of Representativeness: Data science and ML-engineers often work together to tailor optimizations. These optimizations are tested on a small cluster. Often, there is a difference in hardware and load between the test and production clusters, leading to unexpected and novel failures.

Takeaway While infrastructure level is crucial, certain information about the workloads (e.g., performance metrics) are only available at the tenant side. Moreover, while a tenant can try to debug or recreate issues – the hardware differences between production and testing hardware makes this challenging.

4 DESIGN SPACE

The diagnosis challenges faced while training LLMs and DL workloads in public cloud echo the challenges faced while running more traditional systems (e.g. microservices); however, the subtleties discussed in the last section make it more challenging to infer information. Next, we examine the broad set of approaches and highlight specific challenges that limit their applicability. Figure 3 illustrates various levels of cooperation between the tenant and the provider.

Tenant Crowd-sourcing (No Tenant-Provider Cooperation) A promising approach to overcome the lack of information is to cooperate to collect and investigate information

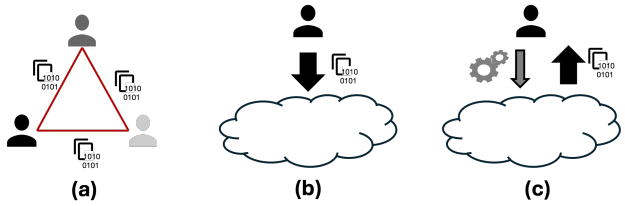

Figure 3: Space of design choices for tenant-provider cooperation for aggregation sufficient information for diagnosis in a public cloud.

on the infrastructure [1]. These approach work in traditional settings as the core information (or signals) being collected are similar. In AI setting, due to diversity in SW and HW, similar signals imply different things depending on the context in which they are collected. Collaboration then requires both sharing signals and key aspects of the workload that introduce privacy concerns.

Hyper-Up Calls (Provider Delegates to Tenant) Another promising approach is to allow tenants to run probes within the provider infrastructure to collect limited information [3], for example, using eBPF. Although this allows for a significant amount of information to be safely exposed, within the AI context the hardware (i.e., infinity band networks and GPU) which provide less opacity and fewer primitives limit the ability to safely employ such techniques. Alternatively, the provider could collect and expose such data. A key challenge here, as with the former, is understanding sufficient context to expose appropriate data.

Provider APIs (Tenant Delegates to Provider) The last approach is to use Provider APIs [2, 6], to leverage provider infrastructure to collect and store appropriate telemetry. This approach enables configurability which tackles the context issues, but still gives providers the freedom to select most efficient methods to collect and potentially share methods.

Our approach builds on the final approach with tenants delegating to the provider. In particular, our system builds on this specific point in the design space for several reasons: first, the opacity of many vendor implementations requires various probing strategies which may be obscured by layers of vulnerability and underlying infrastructure redundancies (e.g., multipath information is nontrivial to infer externally); second, learning and building models requires significant number of runs which will be prohibitively costly; lastly, observability is extremely costly and resource intensive streamlining within the provider results in savings due to reuse.

There are several key challenges: first, as discussed before information about the workloads both software versions, progress reports, and operators are crucial for diagnosis. However, simply sharing this information directly can expose company secrets. Thus, there is a need for a privacy preserving approach to sharing "archetype" information without directly exposing business logic. Even with all the appropriate context, there is a challenge in building statistical models, dealing with novel versions, and minimizing overhead. Next, we discuss our solution and the components.

5 PHAROAH SOLUTION SKETCH

In this work, we assume a public multi-tenant cloud setting in which there is organizational division between the tenants (i.e., data scientists / ML engineers) who write and design the ML-applications and provider (i.e., platform engineers) who owns and manages the cloud infrastructure. As illustrated in Section 3, this division creates a knowledge gap which makes it challenging to fully understand infrastructure components and analyze reliability issues (e.g., job failures). In particular, this gap makes it challenging for a tenant to diagnose memory/load imbalance issues, spot bottlenecks, and estimate resource demands, whereas while the provider can diagnose these issues without context from the tenant's workloads, the provider is unable to tell when such issues are important. While this gulf of information and the challenges it poses when assessing applications deployed in the cloud are well explored in traditional cloud settings [1], the remain under-explored within the context of ML workloads.

As highlighted earlier, a potential solution lies in the reliance of modern AI ecosystem on (i) specialized devices (e.g., GPUs, TPUs) which account for a majority of the error and (ii) common ML-libraries (with limited optimizations). The hardware and software standardization imply that a "bespoke" framework for capturing observability data will generalize across various organizations.

Our system, Pharoah, consists of two phase: An offline phase where benchmarks and historical logs are used to (1) identify "sub-archetypes" and identify hardware centric contexts and (2) learn combinations of "sub-archetypes" and "context" which always lead to failures (e.g., software versions relying on features not supported by hardware) as well as to learn resource usage patterns that more often lead to eventual failures. During the online phase, Pharoah analyzes job metadata at start time to perform proactive patterns based on mismatches, then for jobs without mismatches it continuously monitors to detect signs for gradual degradation patterns that have a high likelihood of leading to eventual failures.

5.1 Offline Training

In offline training phase, Pharoah uses a combination of statistical techniques and machine techniques to detect various workload and hardware characteristics or patterns that correlate highly with failures.

Archetypes identification As others have highlighted [20], the set of computation and communication operators is fixed,

and the use of fixed libraries and optimizations implies that these operators are combined in a fixed set of patterns, we call these fixed patterns "archetypes". One approach to identify the permutations of these operators is to run existing benchmarks [18, 20]. We can also analyze production workloads to infer the specific patterns in which these operators manifest themselves. The former requires access to significant data and an accurate inference tool. The former allows for bootstrapping in the absence of production quality data, e.g., in a new cloud or for new software versions or hardware models. Across various software versions, the characteristics of these operators may vary, which implies that simply focusing on "archetypes" may lead to inaccuracies. To address this, we plan to evaluate different software versions offline and in parallel to further identify clustering of versions within an archetype that behave similarly – referred to as "sub-archetypes".

Context identification While archetypes allow us to extract patterns and variations within the software aspect of AI ecosystem, we require a similar method to identify self-similar hardware and configuration patterns. As other authors [7, 20] have shown, in addition to hardware components, their configuration (e.g., clock speeds or temperature) significantly impacts their failure characteristics [7]. Given our discovery of "sub-archetypes", we benchmarking different combinations of hardware and configuration supported by the cloud provider to identify sets of hardware and configuration combinations, or contexts, which impose similar reliability challenges on AI-workloads.

Learned Failure Patterns Pharoah captures the telemetry data during the benchmarking phase to detect context –Pharoah aggregates the failures information. Pharoah analyzes telemetry after the benchmark is completed and contexts are defined. Given this data, Pharoah tries to extract failure patterns.

Although a certain class of failure can be predicted by evaluating combinations of archetypes and contexts [20] (e.g., mismatch between operators and capabilities exposed by hardware contexts), due to infrastructure-level parallelisms many infrastructure-related errors occur as a result of a build-up over many iterations – essentially AI-workloads degrade gracefully into failures.

Pharoah takes an ensemble approach to tackling these challenges: For the mismatch, Pharoah uses statistical techniques to identify archetypes and contexts that are causality related to failures. For this, we plan to explore causality techniques, e.g., QED. Alternatively for the graceful degradation scenarios we plane to explore survival theory and to learn memory, network, and compute related build-up patterns by injecting various classes of hardware failures while training sub archetypes (Figure 4).

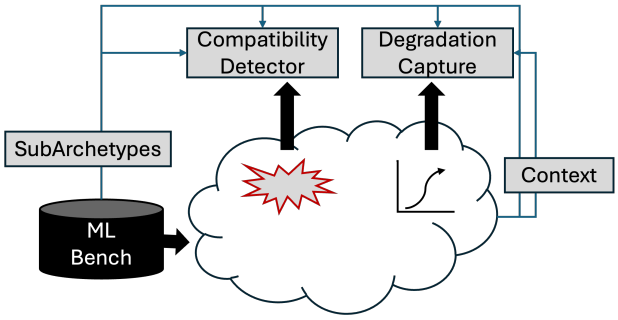

Figure 4: Pharoah's Failure Pattern Learning.

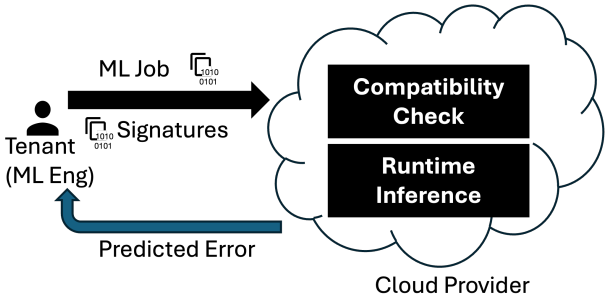

Figure 5: Workflow of Pharoah.

5.2 Online Phase

Online (Figure 5), tenants submit their job and job characteristic in a privacy-preserving manner (i.e., signatures) to Pharoah before starting jobs. Pharoah immediately alerts them to an obvious mismatch between sub-archetypes and hardware contexts. For workloads that do proceed, Pharoah monitors for signs of gradual degradation and when detected confirms with specialized tests.

Privacy Preserving Information Exchange The goal of this project is to determine the root cause of a crash, then given this root cause, to identify components that need to be replaced/avoided or to determine if additional resources are required to make forward progress on a job. As discussed earlier, within the cloud setting, this is challenging because of visibility issues. To help analyze and diagnose challenges that arise due to common components, we introduce a set of signatures, software, and hardware signatures. We envision an offline step which runs various permutations of ML-tool chains to create check sums (or signatures) the potential computation patterns and analyzes the hardware to also create appropriate checksums. These checksums can help identify differences between the training setup and the production setup which could be the cause of performance issues. Ideally, for each signature, our framework learns a recipe of production data to collect to sufficiently understand resource bottleneck and allow for comparisons between runs to detect issues. In an ideal

scenario, such checksums can be used across infrastructures and ML pipelines. A key challenge is minimizing the number of check sums. At one extreme each server configuration can have its own checksum; this would allow for detection of specific combinations that are unfit. Yet, at another extreme, each GPU version can have its own checksum while this is less precise — it may be more appropriate, as most errors are centered on GPUs. With regard to software, checksums can be created on the basis of combinations of versions or combinations of unique code paths. As with hardware, the more precise reduces the potential to learn and generalize, whereas less precise may decrease ability to easily disambiguate.

CrossValidations Tests For the gradual degradation, where failure is uncertain, we run micro-benchmark [12, 20] explicitly designed to test specific resources to determine if infrastructure degradation is in fact occurring.

6 DISCUSSION AND OPEN QUESTIONS

Next, we briefly discuss open research challenges.

How to detect meaningful software change? A key aspect of Pharoah is the discovery of "sub-archetypes". Ideally, every software version presents a candidate for a new "sub-archetype"; however, given the frequency of changes, such an approach does not scale. A more intuitive approach is to explore a combination of program language techniques (e.g. slicing or concolic execution) to identify changes that impact lines associated with realizing operators logic. An orthogonal approach is to use existing benchmarks [18, 20] in new versions to identify differences in operator behavior.

How to identify meaningful information for context? As others have indicated [8, 8, 11, 19], not all configuration options are equally relevant or impact performance equally. At the scale of a large cloud provider [7, 17], we can build on classical feature selection to identify key configuration parameters. We plan to investigate different techniques such as these to reduce the dimensionality of configuration required to define "contexts".

How to bootstrap for newer software and hardware versions? Given the rapid evolution, a challenge lies in learning new context and "sub-archetypes" for emergent hardware or software. We plan to explore ways to short cuts the learning process by running large-scale benchmarks, e.g., SuperBench [20], on emerging components to detect and learn new patterns, contexts, and archetypes.

How to deal with untruthful providers? Intuitively, proactive diagnosis can potentially reduce cloud provider revenues by reducing the amount of time each tenant spends diagnosing problems. However, we note that such a system provides providers with an alternative revenue stream – similar to Google Cloud's Observability [5, 6] or Amazon's CloudWatch [2]. Alternatively, we can explore extensions of cloud-centric privacy-preserving anomaly detection techniques [4] to Pharoah.

7 RELATED WORK

The most closely related work explores a change from manual diagnosis of AI workloads to automated root cause detection and failure detection [12, 20]. Our approach differs in several ways: first, we focus on a public multi-tenant cloud which requires some level of cooperation. Second, we explore a multipronged approach which aims to preemptively detection configuration and placements related failures when possible; however, when not possible, to proactively identify workloads which exhibit patterns of failure. Previous work either takes a reactive approach [20] or aims for a purely preemptive approach [12].

Orthogonal work in this space highlights [9, 20] various sources of resource and energy waste due to inefficiencies, variability, and failures in LLM workloads, recent research of efficiency / reliability is often manual and ad hoc [13, 20] or focused specifically on specific resources (e.g., networks [17]): we lack systematic and automated frameworks to collect and identify sources of inefficiencies and failures in public clouds.

Although there is a rich literature on multi-tenant public clouds diagnosis [10, 15, 16] – these works often focus on traditional workloads. While our work is inspired by the various points in the design space explored by these approaches, our solution is explicitly tailored to the unique characteristics of AI workloads compared with traditional workloads.

8 CONCLUSION

Multi-tenant clouds provide small-medium enterprises and startup with a path toward AI (LLM and DL) training and inference. However, lack of expertise and information about infrastructure often force them to incur additional costs while manually debugging job failures. In this work, we present a vision for automated and proactive detection of failure or preemptive signaling when proactive detection is not possible.

The cornerstone of our approach lies in the standardization of AI software (e.g., pytorch, CUDA) and hardware (e.g, NVIDIA GPUs) stacks, which enables the identification of key patterns that generalize across various proprietary AI-based workloads. We believe that our insight generalizes beyond diagnosis and can enable a broad set of optimizations (e.g., enhancing core libraries will benefit all users) and efficiency algorithms (e.g., designing bespoke tracing frameworks will be broadly applicable across organizations).

REFERENCES

[1] A. Agache, M. Ionescu, and C. Raiciu. Cloudtalk: Enabling distributed application optimisations in public clouds. In G. Alonso, R. Bianchini, and M. Vukolic, editors, *Proceedings of the Twelfth European Conference on Computer Systems, EuroSys 2017, Belgrade, Serbia, April 23-26, 2017*, pages 605–619. ACM, 2017.

[2] Amazon. Amazon cloudwatch. https://aws.amazon.com/cloudwatch/.

[3] N. Amit and M. Wei. The design and implementation of hyperupcalls. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 97–112, Boston, MA, July 2018. USENIX Association.

[4] B. Arzani, S. Ciraci, S. Saroiu, A. Wolman, J. Stokes, G. Outhred, and L. Diwu. {PrivateEye}: Scalable and {Privacy-Preserving} compromise detection in the cloud. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 797–815, 2020.

[5] D. B. G. Cloud. Google stackdriver is now generally available for hybrid cloud monitoring, logging and diagnostics. https://cloud.google.com/blog/products/gcp/google-stackdriver-generally-available//.

[6] G. Cloud. Google cloud's observability. https://cloud.google.com/products/observability?hl=en.

[7] Y. Deng, X. Shi, Z. Jiang, X. Zhang, L. Zhang, Z. Zhang, B. Li, Z. Song, H. Zhu, G. Liu, F. Li, S. Wang, H. Lin, J. Ye, and M. Yu. Minder: Faulty machine detection for large-scale distributed model training, 2024.

[8] S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment*, 2(1):1246–1257, 2009.

[9] A. et al. The llama 3 herd of models, 2024.

[10] V. Harsh, W. Zhou, S. Ashok, R. N. Mysore, B. Godfrey, and S. Banerjee. Murphy: Performance diagnosis of distributed cloud applications. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 438–451, 2023.

[11] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *Cidr*, volume 11, pages 261–272, 2011.

[12] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo, Y. Wen, and T. Zhang. Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 709–729, Santa Clara, CA, Apr. 2024. USENIX Association.

[13] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, Y. Jia, S. He, H. Chen, Z. Bai, Q. Hou, S. Yan, D. Zhou, Y. Sheng, Z. Jiang, H. Xu, H. Wei, Z. Zhang, P. Nie, L. Zou, S. Zhao, L. Xiang, Z. Liu, Z. Li, X. Jia, J. Ye, X. Jin, and X. Liu. Megascale: scaling large language model training to more than 10,000 gpus. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation*, NSDI'24, USA, 2024. USENIX Association.

[14] A. Kokolis, M. Kuchnik, J. Hoffman, A. Kumar, P. Malani, F. Ma, Z. DeVito, S. Sengupta, K. Saladi, and C.-J. Wu. Revisiting reliability in large-scale machine learning research clusters, 2024.

[15] L. Li, X. Zhang, S. He, Y. Kang, H. Zhang, M. Ma, Y. Dang, Z. Xu, S. Rajmohan, Q. Lin, et al. Conan: Diagnosing batch failures for cloud systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 138–149. IEEE, 2023.

[16] H. Nguyen, Z. Shen, Y. Tan, and X. Gu. Fchain: Toward black-box online fault localization for cloud systems. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 21–30. IEEE, 2013.

[17] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, E. Ruan, Z. Yao, E. Zhai, and D. Cai. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 691–706, New York, NY, USA, 2024. Association for Computing Machinery.

[18] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou. Mlperf inference benchmark. *CoRR*, abs/1911.02549, 2019.

[19] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1009–1024. ACM, 2017.

[20] Y. Xiong, Y. Jiang, Z. Yang, L. Qu, G. Zhao, S. Liu, D. Zhong, B. Pinzur, J. Zhang, Y. Wang, J. Jose, H. Pourreza, J. Baxter, K. Datta, P. Ram, L. Melton, J. Chau, P. Cheng, Y. Xiong, and L. Zhou. SuperBench: Improving cloud AI infrastructure reliability with proactive validation. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 835–850, Santa Clara, CA, July 2024. USENIX Association.