Desiderata for next generation of ML model serving

Sherif Akoush [*] [†]
sa@seldon.io

Andrei Paleyes [*] [†] [‡]
ap2169@cam.ac.uk

Arnaud Van Looveren [†]
avl@seldon.io

Clive Cox [†]
cc@seldon.io

Abstract

Inference is a significant part of ML software infrastructure. Despite the variety of inference frameworks available, the field as a whole can be considered in its early days. This position paper puts forth a range of important qualities that next generation of inference platforms should be aiming for. We present our rationale for the importance of each quality, and discuss ways to achieve it in practice. We propose to focus on data-centricity as the overarching design pattern which enables smarter ML system deployment and operation at scale.

1 Introduction

Model inference has become an important part of modern machine learning (ML) infrastructure. Various sources estimate up to 90% of ML compute resources are used for inference tasks [17, 18, 32, 42, 53]. To answer a growing demand for inference infrastructure, a number of model serving platforms appeared on the market [4, 12, 13, 14]. In addition, cloud providers are also offering services that simplify model serving for their users [1, 9].

While the existing ML model serving frameworks answer some of the initial challenges of ML deployment, we believe there are a number of important properties such frameworks need to have to ensure a seamless model deployment experience. For instance, effective monitoring and explainability of an entire inference pipeline is an open research direction. Efficient use of infrastructure while optimising for important metrics of energy and environmental footprint is missing. Seamless ML deployment to different targets such as edge devices is required. While ML serving frameworks are taking steps to provide some of these features, we think that taking into consideration all desired aspects of the system is challenging [41], requires leveraging different architecture patterns, but in general will lead to better solutions developed by the community.

In this position paper we present a set of desired features for ML deployment — a blueprint of the next generation ML model serving frameworks. We discuss motivations for each feature and provide initial pointers on ways to achieve them. We hope to bring the community and practitioners together to discuss these challenges, find ideal solutions, and shape the future of ML serving.

2 Desiderata

In this section we present nine qualities that are important for ML serving. We advocate that designing the system with data-centricity [7, 35] as the highest priority enables these features.

2.1 Inference pipelines as dataflow graphs

Model inference is a complex data processing pipeline. It can include input and output data transformations, multiple ML models, monitoring components, custom business logic, and so on. An

[*] Equal contribution

[†] Seldon Technologies

[‡] Department of Computer Science and Technology, University of Cambridge

NeurIPS 2022 Workshop on Challenges in Deploying and Monitoring Machine Learning Systems.

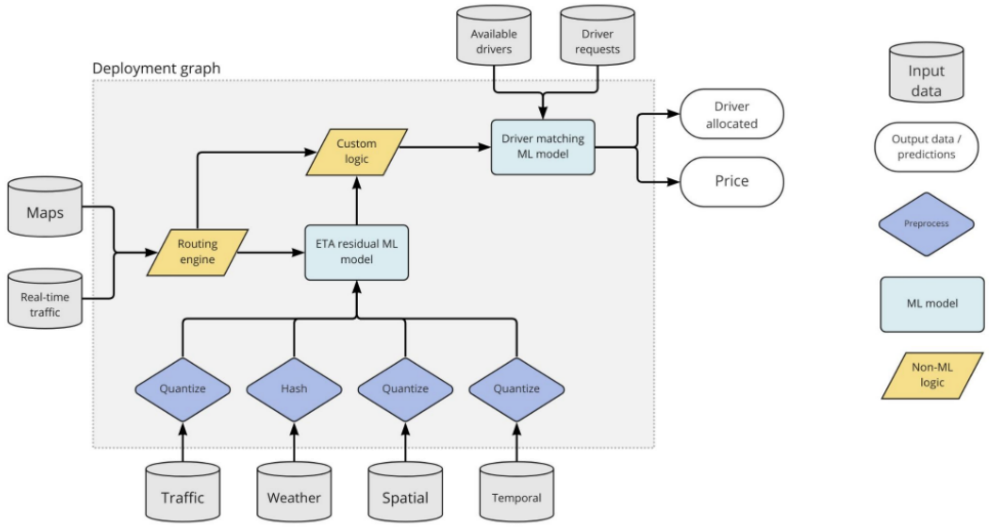

Figure 1: An inference pipeline of a ride-sharing service, motivated by an example from Uber [29]. It ingests multiple data sources, contains several business logic operations and ML models, and produces several outputs.

example of a complex inference graph is shown in Figure 1. It is imperative to have a clear view of the flow of data through the entire pipeline, both for its developers and users. Runtime access to any intermediate data in the pipeline allows for better experimentation, troubleshooting and monitoring experience, all of which are discussed later in this paper.

There are two possible ways for a model serving platform to facilitate access to a dataflow graph at runtime. It can be discovered post-hoc, for example with distributed tracing systems [11, 19]. This approach is applicable to platforms implemented with service oriented approaches, such as microservices. While a popular paradigm for software system design, service orientation might be ill-suited for ML inference, as its control flow nature poorly reflects data-centricity of the inference process. Alternatively, inference pipelines can be built with a dataflow-first approach, such as the flow-based programming (FBP) paradigm, as is already the case for ML training pipelines [20, 34]. FBP provides access to the dataflow graph naturally [40], and thus might be a more appropriate choice for implementation of inference graphs.

2.2 Pipeline component abstractions

To allow for the construction of complex data pipelines as shown in Figure 1, simple but flexible architectural building blocks are required. ML models usually run on dedicated specialised servers. The models will have generally been created by data scientists while the final serving infrastructure is usually handled by a separate dedicated operations teams. The definition of models and servers for inference should be kept separate to allow for their distinct creation and for the possibility of model sharing across servers.

Given a set of models, many data pipelines may be built which share them at inference time. Consequently, a pipeline abstraction should be a higher level concept that defines the flow of data between the functional steps and how that data is joined and split as needed. Teams should be able to tap into any data source, consume any output stream or extend the inference pipeline for extra processing.

2.3 Support for synchronous and asynchronous scenarios

Modern ML inference services should natively support two modes of operation: synchronous and asynchronous. Synchronous inference is the traditional mode of doing inference, also known as request-driven batch processing. In this mode a user makes a request with a batch of input data points to the platform, and receives a response with corresponding predictions. For that interaction to happen the inference platform has to provide an API endpoint suitable for accepting prediction requests [49]. Asynchronous inference allows for a different interaction pattern, where the input data is arriving and the predictions are produced continuously. Such behaviour can be achieved with data streams, and is often referred to as stream processing [46].

With real-time ML gaining attention because of its data availability, flexibility and ability to scale [30], asynchronous use cases will be encountered more often. Stream processing possesses a range of qualities which make it better suited for real-time analytics [52], such as the ability to adapt to variable workloads [24, 33]. At the same time, reports estimate around 77% of software development teams are working with service-based architectures [39], which means it is easier for many users to include batch processing APIs in their software setup. Remarkably, hybrid approaches are now being proposed, in an attempt to provide convenient interfaces for both synchronous and asynchronous interaction modes [22, 25, 26].

2.4 Seamless deployment to Cloud and Edge

ML models are trained in the cloud using a big fleet of powerful machines and potentially deployed to many different hardware for inference. For example an image recognition model can be deployed to cameras for online object detection, and the same model can be hosted in a data centre for background feed processing. Even within the same data centre the same ML model might be deployed to different machine specs with or without hardware accelerators.

One of the challenges is to optimise the model performance according to the target hardware. There are existing tools that can help automatically generate target-specific artefacts employing techniques such as ML model quantisation, operator fusion and network pruning [2, 17, 27, 36, 43]. However it is a complex multidimensional optimisation problem that should take into consideration cost, latency and throughput targets.

Therefore the ML serving framework should take care of setting up the infrastructure as required, optimising the inference service accordingly. ML serving frameworks shall be able to provide the same set of features to the users, regardless of where models are deployed.

2.5 Flexible experimentation

ML is an iterative task where new components are updated over time to better harness the data that is currently flowing through them. Therefore, an inference system needs flexible and simple ways to test updates both to individual atomic models as well as entire data pipelines. Furthermore, as the number of data pipelines increases the task of validating experiments manually becomes more onerous. In recent years the use of progressive rollouts for new models where clear service level agreements (SLAs) and acceptance criteria are defined and the data traffic is managed automatically between challenger and champion models is becoming more popular with tools on the market [3, 8, 10]. These techniques need to be extended to handle entire data pipeline rollouts in an efficient manner. Managing experiments at scale also requires smarter monitoring which we describe next.

2.6 Monitoring and explainability

The key responsibility of monitoring is to identify abnormal situations in the inference system and allow fast and accurate root-cause analysis (RCA). A good monitoring solution for ML systems needs to be able to flexibly allow RCA for any connected set of components in the inference graph. Access to the dataflow graph at runtime, as discussed in Section 2.1, provides the necessary intermediate data streams and runtime graph traversal operations to achieve such flexibility.

A particular example of such a flexible monitoring feature is context-aware drift detection [23], where a data drift observed downstream can be analysed jointly with some of the upstream data used as a context. Another example is recursive attribution [47], where an observed behaviour is linked to the most likely cause, and this procedure is repeated recursively until the user input is reached.

Operations and auditing teams need to be able to explain on demand any part of a pipeline for technical, regulatory or business reasons [31]. When data snapshots itself are not sufficient to derive such explanations, inference systems should allow to dynamically add ML interpretation models.

2.7 Continual and active learning

ML model performance in production degrades over time. To address this challenge the first step is to detect performance issues via in-depth RCA as described in Section 2.6. The next step is to trigger (re)training of the problematic ML model on new data, compare metrics between the existing and candidate model, and automatically rollout the new version to production. Therefore ML serving frameworks should integrate well with ML training tools, exposing data from production that can improve the (re)training process. An extension is to support active learning [45] techniques in which there is a tight integration between data acquisition, incremental (re)training and inference.

2.8 Data privacy and compliance

With more ML-driven solutions used to automate sensitive decision making, ML applications become governed by various legislatures, such as GDPR in Europe, PIPEDA in Canada or APPI in Japan. Users that run their workloads on model inference platforms have to protect privacy of the data passing through the pipeline, prove their compliance with relevant regulations, and provide explanations of decisions made by their models.

Consequently, it is important for ML inference frameworks to aid compliance and provide privacy guarantees. Access to the complete dataflow graph can be a reasonable way to achieve these requirements, as it can facilitate privacy assessment [48] and enable "compliance by construction" [44]. In particular, framework users need to be able to answer questions such as *What are the assumptions behind the observed output? What are the main input factors which contributed the most to the output? What minimal changes to the input can change the output?*. Collectively such meta information about data is known as data provenance [21], and the ability to systematically answer these questions is a critical mechanism for assuring compliance.

Literature describes multiple attacks against deployed models, such as model inversion [51] and model stealing [50], mostly targeted at recovering parts of private training datasets and model parameters. Defence methods against such attacks are being actively explored [37], and should be considered as a part of any modern model inference platform.

2.9 Resource efficiency

Detailed cost and energy metrics. Doing inference over the lifetime of an ML model is energy hungry. Inference can reach up to 90% of the energy consumed during the ML model lifecycle [38, 42, 53]. While this adds costs to organisations, as ML becomes integrated in our daily lives we should strive to have a holistic accounting of ML energy use that incorporates all related tasks [28]. The community is already taking steps to address some of these challenges and account for energy and CO_2 emissions of ML training tasks [5, 42, 53]. These techniques should be extended to ML serving.

Multimodel serving with overcommit of resources. We advocate for multimodel serving pattern where one ML inference server hosts multiple models at the same time. This reduces overheads required to deploy a large number of models while making it simpler to operate the system at scale (check Appendix A for more details). Multimodel serving is already a feature provided by some inference servers and frameworks [12, 15, 16]. Moreover, in many cases demand patterns allow for further optimisation such as overcommit of resources. This means that an ML system could register more models than what can be served by the provisioned infrastructure. The system should be able to swap models dynamically according to usage without adding significant latency overheads to inference requests. A complementary approach is autoscaling of resources (e.g. replicas of inference nodes) according to load.

3 Conclusion

With this position paper we hope to reinforce the importance of research and development on ML inference systems and encourage a debate on the desired requirements for the next generation of tools to successfully deploy and manage powerful predictive pipelines.

References

[1] Amazon SageMaker. Available at https://aws.amazon.com/pm/sagemaker.

[2] Apache TVM. Available at https://tvm.apache.org/.

[3] Argo rollouts: Kubernetes progressive delivery controller. Available at https://argoproj.github.io/argo-rollouts/.

[4] Bento ML. Available at https://github.com/bentoml/BentoML.

[5] CodeCarbon. Available at https://codecarbon.io/.

[6] Considerations for large clusters. Available at https://kubernetes.io/docs/setup/best-practices/cluster-large/.

[7] Data-centric AI Resource Hub. Available at https://datacentricai.org/.

[8] Flagger: Progressive delivery operator for kubernetes. Available at https://flagger.app/.

[9] Google Vertex AI. Available at https://cloud.google.com/vertex-ai.

[10] Iter8: Kubernetes release optimizer. Available at https://iter8.tools/.

[11] Jaeger: open source, end-to-end distributed tracing. Available at https://www.jaegertracing.io/.

[12] KServe. Available at https://github.com/kserve/kserve.

[13] Ray. Available at https://github.com/ray-project/ray.

[14] Seldon Core. Available at https://github.com/SeldonIO/seldon-core.

[15] Seldon MLServer. Available at https://github.com/SeldonIO/MLServer/.

[16] Triton Inference Server. Available at https://github.com/triton-inference-server/server.

[17] R. Y. Aminabadi, S. Rajbhandari, M. Zhang, A. A. Awan, C. Li, D. Li, E. Zheng, J. Rasley, S. Smith, O. Ruwase, and Y. He. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale, 2022.

[18] J. Barr. Amazon EC2 update – Inf1 instances with AWS Inferentia chips for high performance cost-effective inferencing, 2019. Available at https://aws.amazon.com/blogs/aws/amazon-ec2-update-inf1-instances-with-aws-inferentia-chips-for-high-performance-cost-effective-inferencing/.

[19] J. Berg, F. Ruffy, K. Nguyen, N. Yang, T. Kim, A. Sivaraman, R. Netravali, and S. Narayana. Snicket: Query-driven distributed tracing. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, pages 206–212, 2021.

[20] S. Cai, E. Breck, E. Nielsen, M. Salib, and D. Sculley. Tensorflow debugger: Debugging dataflow graphs for machine learning. 2016.

[21] L. Carata, S. Akoush, N. Balakrishnan, T. Bytheway, R. Sohan, M. Seltzer, and A. Hopper. A primer on provenance: Better understanding of data requires tracking its history and context. *Queue*, 12(3):10–23, 2014.

[22] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[23] O. Cobb and A. Van Looveren. Context-aware drift detection. In *International Conference on Machine Learning*, pages 4087–4111. PMLR, 2022.

[24] T. Das, Y. Zhong, I. Stoica, and S. Shenker. Adaptive stream processing using dynamic batch sizing. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, page 1–13, New York, NY, USA, 2014. Association for Computing Machinery.

[25] D. Dissanayake and K. Jayasena. A cloud platform for big IoT data analytics by combining batch and stream processing technologies. In *2017 National Information Technology Conference (NITC)*, pages 40–45. IEEE, 2017.

[26] A. Fino, A. Margara, G. Cugola, M. Donadoni, and E. Morassutto. RStream: Simple and efficient batch and stream processing at scale. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 2764–2774. IEEE, 2021.

[27] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[28] A. Hopper and A. Rice. Computing for the future of the planet. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3685–3697, 2008.

[29] X. Hu, T. Binaykiya, E. Frank, and O. Cirit. DeeprETA: An ETA post-processing system at scale. *arXiv preprint arXiv:2206.02127*, 2022.

[30] C. Huyen. Machine learning is going real-time, 2020. Available at `https://huyenchip.com/2020/12/27/real-time-machine-learning.html`.

[31] J. Klaise, A. Van Looveren, C. Cox, G. Vacanti, and A. Coca. Monitoring and explainability of models in production. *Workshop on Challenges in Deploying and Monitoring Machine Learning Systems, ICML*, 2020.

[32] G. Leopold. AWS to offer Nvidia's T4 GPUs for AI inferencing, 2019. Available at `https://www.hpcwire.com/2019/03/19/aws-upgrades-its-gpu-backed-ai-inference-platform`.

[33] B. Lohrmann, P. Janacik, and O. Kao. Elastic stream processing with latency guarantees. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 399–410. IEEE, 2015.

[34] T. Mahapatra and S. N. Banoo. Flow-based programming for machine learning. *Future Internet*, 14(2):58, 2022.

[35] L. J. Miranda. Towards data-centric machine learning: a short review. *ljvmiranda921.github.io*, 2021.

[36] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort. A white paper on neural network quantization, 2021.

[37] D. Oliynyk, R. Mayer, and A. Rauber. I know what you trained last summer: A survey on stealing machine learning models and defences. *arXiv preprint arXiv:2206.08451*, 2022.

[38] Open AI. AI and Compute, 2018. Available at `https://openai.com/blog/ai-and-compute/`.

[39] O'Reilly. Microservices adoption in 2020: a survey, 2020. Available at `https://www.oreilly.com/radar/microservices-adoption-in-2020/`.

[40] A. Paleyes, C. Cabrera, and N. D. Lawrence. An empirical evaluation of flow based programming in the machine learning deployment context. In *2022 IEEE/ACM 1st International Conference on AI Engineering – Software Engineering for AI (CAIN)*, pages 54–64, 2022.

[41] A. Paleyes, R.-G. Urma, and N. D. Lawrence. Challenges in deploying machine learning: A survey of case studies. *ACM Comput. Surv.*, apr 2022. Just Accepted.

[42] D. A. Patterson, J. Gonzalez, Q. V. Le, C. Liang, L. Munguia, D. Rothchild, D. R. So, M. Texier, and J. Dean. Carbon emissions and large neural network training. *CoRR*, abs/2104.10350, 2021.

[43] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean. Efficiently scaling transformer inference, 2022.

[44] M. Schwarzkopf, E. Kohler, M. Frans Kaashoek, and R. Morris. Position: GDPR compliance by construction. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 39–53. Springer, 2019.

[45] B. Settles. Active learning literature survey. 2009.

[46] S. Shahrivari. Beyond batch processing: towards real-time and streaming big data. *Computers*, 3(4):117–129, 2014.

[47] R. Singal, G. Michailidis, and H. Ng. Flow-based attribution in graphical models: A recursive Shapley approach. In *International Conference on Machine Learning*, pages 9733–9743. PMLR, 2021.

[48] F. Tang and B. M. Østvold. Assessing software privacy using the privacy flow-graph. *arXiv preprint arXiv:2209.02948*, 2022.

[49] A. Team. AzureML: Anatomy of a machine learning service. In L. Dorard, M. D. Reid, and F. J. Martin, editors, *Proceedings of The 2nd International Conference on Predictive APIs and Apps*, volume 50 of *Proceedings of Machine Learning Research*, pages 1–13, Sydney, Australia, 06–07 Aug 2016. PMLR.

[50] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

[51] M. Veale, R. Binns, and L. Edwards. Algorithms that remember: model inversion attacks and data protection law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2133):20180083, 2018.

[52] W. Wingerath, F. Gessert, S. Friedrich, and N. Ritter. Real-time stream processing for big data. *it-Information Technology*, 58(4):186–194, 2016.

[53] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. Aga, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H.-H. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. Hazelwood. Sustainable AI: Environmental implications, challenges and opportunities. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 795–813, 2022.

Appendix A Multimodel vs single model serving

For serving ML models in production, a standard pattern is to package up the ML model inside a container image which then gets deployed and managed by a service orchestration framework (e.g. Kubernetes). A slight variation of this pattern is to have the ML model persisted separately (e.g. in an object store) and the orchestration framework fetching and injecting the model artifact inside the container runtime during startup. While this pattern works well for organisations in the case of deploying a couple of models, it does not scale well as there is a one-to-one mapping between a deployed container and an ML model being served. With the requirement to deploy many thousands of models in production there is going to be a lot of extra resource overhead to keep these containers running in the system.

To illustrate this concept with a working example there is a current Kubernetes limitation on the number of pods per node, which is 110 pods per node [6]. To deploy 20,000 single pod ML models we would need then 200 nodes. The smallest node in Google Cloud is `e2-micro` (1 GB memory) and therefore the total system would require at least 200 GB of provisioned memory. In fact the memory requirement is likely to be far greater as it is not possible to have 100s of model inference server pods on a small node. With multimodel serving however the memory footprint of the system is expected to be one order of magnitude less by design as resources are shared at the model level.

Multimodel serving has also additional benefits. It allows for better CPU/GPU sharing. It does not suffer from the issue of cold start, where with each ML model to deploy we have to download the container image before starting it — this is usually in the order of tens of minutes. Multimodel serving also reduces the risk of allocating new cloud resource (e.g. GPU) on-demand as model inference servers are long-lived by design.