ModelOps: Cloud-based Lifecycle Management for Reliable and Trusted AI

Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, Kaoutar El Maghraoui
IBM Research AI
1101 Kitchawan Rd, Yorktown Heights, NY 10598, United States
{whummer,thomas.rausch}@ibm.com, {vmuthus,pdube,kelmaghr}@us.ibm.com

***Abstract*—This paper proposes a cloud-based framework and platform for end-to-end development and lifecycle management of artificial intelligence (AI) applications. We build on our previous work on platform-level support for cloud-managed deep learning services, and show how the principles of software lifecycle management can be leveraged and extended to enable automation, trust, reliability, traceability, quality control, and reproducibility of AI pipelines. Based on a discussion of use cases and current challenges, we describe a framework for managing AI application lifecycles and its key components. We also show concrete examples that illustrate how this framework enables managing and executing model training and continuous learning pipelines while infusing trusted AI principles.**

***Index Terms*—model, pipelines, DevOps, AI, cloud, harden, monitor-drift**

I. Introduction

Advances in artificial neural networks, coupled with the explosive growth of data volumes and advances in processing power, have led to rapid adoption of deep learning in a variety of applications [1]. Along with algorithmic breakthroughs, the community is increasingly focused on system and process level aspects to support scalable machine learning on massive data sets. Major Cloud providers are also racing to provide a comprehensive stack that delivers artificial intelligence (AI) as a service (e.g., Amazon, IBM, Microsoft and Google) by offering clusters of GPU-backed VMs and containers through a pay-as-you-go pricing model. AI application developers can therefore leverage the cloud computing infrastructure to train machine learning models and take advantage of additional compute services such as batch processing, container orchestration and serverless computing for parallelizing and automating machine learning workloads.

Additionally, cloud platforms lend themselves naturally to AI pipelines that tend to be bursty and require massive experimentations (e.g., hyperparameter optimization in Deep Learning tasks). Cloud is also playing a key role in bringing AI to the masses and democratizing its uses among a variety of AI practitioners, by removing the necessity of owning expensive infrastructure to start experimenting with AI techniques.

However, systematic lifecycle support—including continuous development, training, testing, and deployment of models—and continuous integration [2] for AI is still in its infancy. AI has shown promise in a variety of application domains but its adoption by enterprises is still nascent. We think one reason for this is the lack of tools and methodologies to support the development lifecycles of AI solutions, spanning data preparation, model design and training, application development, quality checks (e.g., security, bias, compliance, etc.), deployment, monitoring, and feedback, as well as the reproducibility and auditability of the entire process. Machine learning teams typically build custom tools and work in ad hoc ways to address requirements in different parts of the AI lifecycle. This slows down the operationalization of AI applications and is at odds with the business needs to rapidly deploy models and continuously evaluate the quality of them in a production setting. There is clearly a need for a more streamlined and systematic approach to AI application development and lifecycle management.

Established practices in cloud software engineering, such as continuous integration (CI), continuous delivery (CD), and staged deployments, have brought tremendous benefits such as scalability of projects to large teams, productivity of developers, reproducibility of errors, and ability to deliver code quickly with minimal risk. Applying these principles to AI applications will bring similar benefits. We hypothesize that, by leveraging concepts from software engineering and DevOps [3], [4], we can accelerate the AI lifecycle, foster collaboration and reuse of AI models across teams with diverse skill sets, and ultimately achieve higher quality AI applications.

This paper introduces ModelOps (a portmanteau of AI models and DevOps), a novel framework and platform for end-to-end lifecycle management of AI application artifacts. We build on our previous work on a cloud-native model training platform [5], [6], and show how to leverage and extend the principles of software lifecycle to co-evolve AI models and the applications they are deployed in. While AI lifecycles can be structurally similar to traditional application lifecycles, we show that there are some fundamental differences which bring new challenges and require changes to traditional DevOps pipelines.

One of the key components of ModelOps is a domain abstraction language with first-class support for the common artifacts in AI solutions. This includes datasets, model definitions, trained models, applications, and monitoring events, as well as the algorithms and platforms used to process data, train models, or deploy applications. These artifacts are all

#	Domain	Model Types	# Models	Retraining	Key Focus and Characteristics
UC1	Retail	Price Optimization	10s	semi-automated	Model variants for geographical regions
UC2	Health Care	Time Series	1000s	manual	Patient-specific models; HIPAA compliance
UC3	Device Maintenance	Visual Recognition	10s	semi-automated	Retraining loop based on user feedback
UC4	Financial Services	Time Series	100s	manual	Stringent rules for bias, auditability
UC5	Conversation	NLP/NLU	10s	semi-automated	Ad-hoc pipelines to generate entities/intents
UC6	ML Competitions	Various	100s	automated	Framework for solving Kaggle competitions

TABLE I
AI APPLICATION USE CASES

versioned, and their lineage is tracked for reproducibility and auditability. The abstractions are defined at a level of granularity that makes it possible to easily author pipelines customized for a team's development workflow.

Another important aspect of ModelOps is flexibility. We have seen that teams have preferences in terms of the tools and (cloud) platforms they use, and ModelOps is designed to meet the teams where they are. Proprietary model deployment platforms or custom data processing services can be easily plugged into a ModelOps pipeline. ModelOps also allows easily infusing quality control checks in the lifecycle of an AI application. Through its pluggable design, users can easily bootstrap AI pipelines that infuse security checks [7], bias checks [8], explainability [9], compliance checks, or any other quality controls and their mediations that are deemed necessary to deploy AI in enterprise settings.

The rest of the paper is structured as follows. In Section II we first discuss the unique properties of AI application lifecycle and how they differ from traditional applications lifecycles, and outline some key uses cases that motivate the need for ModelOps. In Section III, we discuss the design principles of ModelOps and sketch the system architecture and its components. Section IV describes selected implementation details of the proposed framework. Section V discusses relevant related work and how it compares with our approach. The paper ends with a summary of the benefits, conclusion and future work in Section VI.

II. OPERATIONALIZING AI

For AI to become mainstream, it will need to move beyond small-scale and often ad-hoc experiments run by data scientists to automated operationalized pipelines with inference and results being directly consumed in enterprise settings. Operationalizing AI implies integrating AI models and algorithms in production applications, and encompasses properties on how the models behave and evolve, and the processes teams use to build, quality control, and consume these models.

To better understand the requirements and challenges teams face while operationalizing AI, we have reviewed several use cases and conducted interviews with internal and external teams working on research as well as industry projects involving AI. This section presents a classification of the key requirements that arise in the use cases. The section also highlights characteristics of AI lifecycles that differ from traditional application lifecycles, as well as a summary of the challenges that need to be addressed by any platform that supports these use cases.

A. Use Cases and Challenges for AI Lifecycles

Table I summarizes a set of use cases (UC1-UC6) and their key characteristics. Our analysis and interviews capture (1) general information about the use case and personas involved, (2) model retraining and automation requirements, (3) management of model metadata and lifecycle events, and (4) promotion of models across environments (e.g., dev/staging/prod). Although a survey of this size is not statistically representative, we are still able to identify interesting challenges and pain points faced by users, which further motivated our work. Below are some of the key insights:

Automation: Automating the entire pipeline from data preparation through model training and deployment to runtime monitoring can be time consuming and error-prone. Particularly UC1 and UC6 reported that teams find themselves creating lots of boilerplate code, instead of focusing on the core data science and machine learning algorithms.

Quality Assurance: As AI models take a more central role in enterprise applications, steering critical decisions like credit worthiness checks (UC4) or health monitoring (UC2), the issues of model fairness and robustness become pressing. Users must be able to easily plug in quality assurance checkers (e.g., bias checks, drift detection, etc.) to continuously assert the performance and reliability of their models.

Traceability: Teams need to be able to answer questions like *"Which data was this model trained on?"*, or *"Which code or data change made our accuracy deteriorate?"*. However, keeping track of data and models across the lifecycle is difficult with little tooling support. This is required in regulated domains like health care or finance (UC2, UC4).

Risk Management: Rolling out new model versions in an application introduces risks, and teams need to manage the risk. Possible techniques include canary releases, A/B testing with user feedback, and drift detection. These are essential features for use cases UC1, UC3, and UC4.

Feedback Cycle: Continuous improvement of models requires an efficient feedback loop all the way from the user interface to the model training backend. Instead of handcrafting all steps in this cycle for each use case, teams are demanding reusable patterns and tools that optionally allow humans to be involved in the loop. This requirement is strongly driven from use cases UC3 and UC5.

B. Characteristics of AI Application Lifecycles

While AI lifecycles can be structurally similar to traditional application lifecycles, there are qualitative and quantitative differences (see Table II).

AI can require a more diverse set of skills (e.g., data science, statistics). Also, the workflow stages can be much longer (e.g., data preparation), and be executed more frequently (e.g., continuous training). Also, AI applications often need to manage a huge number of artifact versions (e.g., models trained with different configuration parameters or datasets). Model artifacts also do not necessarily evolve in a linear fashion with incrementing version numbers; many model variants may coexist, including those personalized for subsets of users. Finally, while classical applications adapt and apply configurations at runtime, AI apps require computationally intensive algorithms at training time.

When it comes to unit and integration tests, traditional testing typically runs on a small set of inputs and expects to generate stable and reproducible results. However, as the domain of AI models is stochastic in nature, test results may not be (exactly) reproducible. Another key difference is around build triggers - in addition to code changes, any (significant) changes in the data assets may require to re-build and evaluate the quality of the AI solution. Changing one thing might lead to changing everything in the AI world.

C. Experience from the Front Line

Watson Machine Learning (WML) [10] is a cloud platform enabling users to train, manage, and deploy models using an automated, collaborative workflow. Our research on ModelOps builds on insights we have gathered from analyzing how WML is used in both production and research settings.

With the basic workflow automation that WML provides today, a large portion of users already deploy models multiple times a day. We found that, users who use the system regularly deploy on average between 1.2 to 226 times per day, with some accounts peaking at 1000 daily deployments, clearly showing the need for scalable systems to enable end-to-end automation of AI workflows. Compared to classical DevOps CI/CD workflows, AI pipelines can be very long running and their execution time varies greatly depending on pipeline complexity and the frameworks used. For example, the average pipeline for simple models runs around 20 minutes, where 7% is spent on data preprocessing, 89% is on training, and 4% on validating the model. A typical long-running pipeline may run for more than 13 hours, and some deep learning jobs can even run up to several days. Similar observations for frequency and duration of different training workloads are also reported by Facebook applying AI to their workloads [11].

We also identify the importance for AI operations platforms to abstract from the libraries and frameworks used. For example, WML lets users chose the learning frameworks for their pipeline. Some 63% of submitted training jobs use Spark, 32% TensorFlow, 3% PyTorch, and 1% use Caffe. The others are a mix of Darknet, MXNet, or Theano jobs. Furthermore, we found that, while 81% of data processing flows operate on IBM's Cloud Object Storage, almost 18% read and write data from various relational databases.

D. AI Operations Platform Challenges

This section discusses the challenges in building scalable AI operations. These serve as requirements for the subsequent sections on the design and implementation of ModelOps.

Pluggability: Lifecycle services must be easily pluggable and customizable. Consuming services like bias or robustness checks in the lifecycle still comes with a relatively high cost of custom integration, as reported by UC4/UC5. Additionally, some AI pipelines still require human in the loop such as continuous learning pipelines where a human needs to validate certain outcomes or re-label some sample data for re-training. Therefore, we need to support pipelines that are completely automated and others that still require a human in the loop interventions.

Reusability: To provide an easy on-ramp and to keep the configuration effort at a minimum, users should be able to use pre-configured templates and patterns for pipelines and lifecycle capabilities.

Flexibility: The system needs to meet developers and data scientists where they are, and integrate with the tools and services they already use and are comfortable with.

Scalability: The system needs to provide scalability across different dimensions - including model metadata versioning, parallel pipeline executions, event processing, and model performance monitoring.

Hybrid Environments: While there is a general move to the cloud, a significant number of AI systems use on-premise servers, dedicated clusters, edge devices, or a combination of these. Resource and security heterogeneity in such hybrid environments introduce a number of challenges.

Fault Tolerance: As ModelOps pipelines plug together a wide range of tools and infrastructure, many things can fail. For example, we found that 8% of all data preprocessing tasks in WML fail or finish with errors, and therefore would stall the subsequent training of a model. This is particularly problematic for automated pipelines of critical models.

III. System Design

Based on the identified challenges, we develop ModelOps, a system to operationalize AI. ModelOps revolves around the core concept of pipelines: a series of tasks that generate, monitor, and continuously improve AI models. We introduce a metamodel that captures concepts necessary to develop, generate, and execute pipelines. Furthermore, ModelOps introduces production stages and environments as top-level concepts. Figure 1 illustrates the coarse-grained architecture, including the internal component stack and external plugin connectors.

A. Model Training Pipelines

We use the concept of *pipelines* to express the logic applied in complex automated model training workflows. Users express pipelines as a directed acyclic graph (DAG) where each node represents a task and each edge defines the control flow

Classical Application Lifecycle	AI Application Lifecycle
Requires dev / ops skills	Involves more diverse skill sets
Relatively short running	Long-running, resource intensive
Human speed (low change frequency)	Continuous (re-)training
Few versions of software artifacts	Huge number of models
Linear evolution of artifacts	Specialized models coexist
Configurations applied at runtime	Parameters tuned at training time
Codebase changes trigger new builds	Data/code changes trigger new builds (re-training)
Deterministic testing	Statistical/probabilistic testing
Monitoring of application performance and KPIs	Monitoring of model accuracy, drift, and KPIs

TABLE II
COMPARISON OF APPLICATION DEVELOPMENT LIFECYCLES

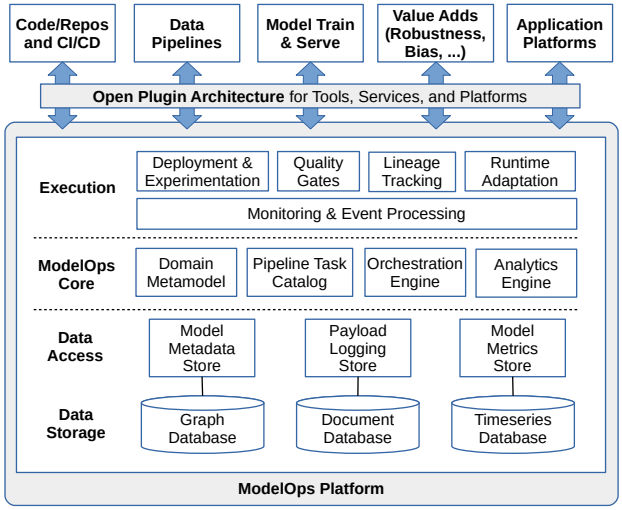

Fig. 1. ModelOps Platform Architecture

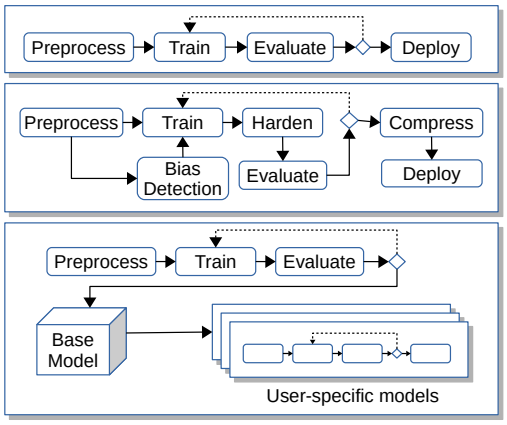

Fig. 2. Sample Pipelines: (1) Simple train/deploy pipeline, (2) complex pipeline with custom plugins enabled, (3) transfer learning pipeline with user-specific models

between tasks (including branch and join nodes). Typical tasks include data preprocessing, training and deployment, but also extend to more advanced techniques such as model hardening [12] and [7], compression [13], and bias mitigation [14]. To allow for cyclic functionality (e.g., model retraining loops), ModelOps introduces the concept of events and triggers (see next subsection).

As ModelOps is part of our ongoing effort to democratize AI, it is important to make composition and operation of pipelines accessible to a broad range of users. While the DAG structure and plugin support for executing arbitrary code as tasks provide the necessary flexibility to implement complex custom pipelines, we found that a set of prototypical structures cover a large set of common use cases.

The use of well defined pipelines primarily solves the Automation requirements of virtually every use case we have come across. Figure 2 shows different pipeline prototypes that we have identified from the use cases outlined in the previous section, and the tasks used to implement the pipelines. Pipelines can also support complex automated Quality Assurance checks required by many use cases.

B. AI Operations Metamodel & Metadata Management

ModelOps tracks metadata across the lifecycle, and defines metamodel entities that abstract the core concepts of AI operations: assets (including training data sources and models), accounts (including access keys for data sources), pipelines and tasks, and event triggers. All entities are automatically versioned and the full history can be queried.

Figure 3 illustrates a simple timeline with events and tasks for model training and hardening, as well as the entities tracked throughout the process. At time t_1 a monitoring process detects that the training data has changed, and a corresponding event is raised. This event triggers a *Train Model* task which retrains the model m_1 and then raises a *Model Updated* event at t_2. Since the CLEVER score [15] of this model is relatively low (0.721), a *Harden Model* task is triggered which tests and hardens the model against various attacks and generates a new model at time t_3, with improved CLEVER score of 0.803.

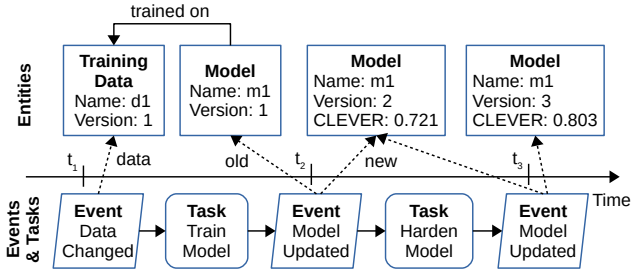

Fig. 3. Simple Model Training Pipeline with Metadata

Incorporating metadata management at the core of ModelOps allows us to answer the Traceability questions raised by several use cases as described in the previous section.

C. Monitoring and Event Triggers

What distinguishes AI pipelines from classical DevOps workflows is the notion of closed feedback loops between the deployed artifacts and the pipelines that generate them. After a model has been deployed, and it is used by an application, runtime monitoring data on the model are fed back into the pipeline to drive decisions on when run specific pipelines. For example, UC2 requires models to be retrained when a certain number of new labeled data are available, and if the performance of the model drops below a given threshold. In UC4, bias detection algorithms are used to monitor bias of a deployed model, and may trigger the execution of a pipeline to harden that model. In ModelOps, such rules can be formulated as *event triggers*. To operate these rules, ModelOps monitors the runtime performance of deployed models, as well as data store metrics.

Events allow for the runtime monitoring of models and applications, and are needed for the Feedback Cycle required by some use cases. Events can invoke pipelines to act on runtime signals, completing the feedback loop.

D. Environment Abstractions

A common use case for AI operations is to promote data, model, or application assets between environments or operational stages. After models are developed in a *local* environment, it moves through a *staging* phase into *production*. Because they are crucial in enabling flexible development workflows, ModelOps treats environments as first-class citizens in the metamodel. Instead of duplicating configurations for pipelines and tasks, configurations are parameterized with environment specific overrides. Promotion of assets between environments can be automated, or may require sign-off from a human in the loop, which is recorded in the metadata for auditability purposes.

The concept of environments as a first class artifacts was introduced in response to requests from several teams, and allows users to easily encode best practices to address the Risk Management concerns.

E. Cross-Cloud Model Training Pipelines

Another key requirement for operationalizing AI is to provide cross-platform abstractions for the key lifecycle capabilities. For example, even though the APIs for Amazon's AWS SageMaker[1] and IBM's Watson Machine Learning[2] (WML) platforms provide similar capabilities, the underlying programming model and API interfaces show some significant differences. Whereas SageMaker generally requires the user to provide a custom Docker image with pre-defined entry points, WML allows the user to upload a zip archive with the model training code, which then gets deployed into a container in the cloud environment. Our goal in ModelOps is to abstract from these subtle API differences, allowing the user to focus on the data science rather than having to deal with the low-level API calls and configuration management.

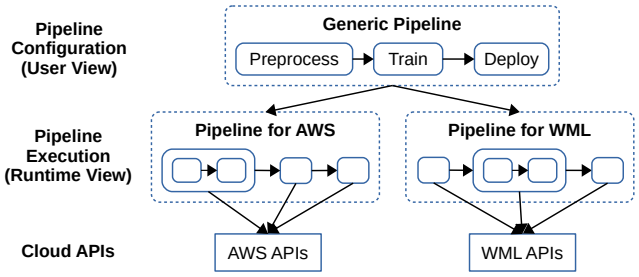

Fig. 4. Mapping Generic Pipelines to Cloud Platform Specific APIs

Figure 4 illustrates how ModelOps allows to seamlessly switch between execution environments. The generic pipeline definition specified by the user is independent of the target platform. Upon execution, the runtime automatically instantiates the corresponding task implementations, depending on which target cloud platform has been specified (e.g, AWS or WML).

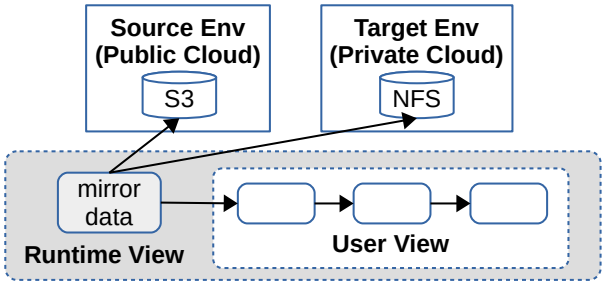

Fig. 5. Injecting Preprocessing Tasks into the Pipeline

The notion of generic pipelines also allows to inject preprocessing tasks that facilitate the switching between environments. Consider the scenario in Figure 5 which illustrates a generic pipeline (User View) to train a model in a private cloud environment, using a network file system (NFS) for data storage. The user's training data happens to reside in an S3 bucket of a public cloud, therefore during execution (Runtime View) we automatically inject a task into the pipeline which mirrors data from the source bucket to the NFS storage, thereby making it available to the target environment.

IV. Implementation

A. Declarative Pipeline Configuration

Executing AI pipelines requires access to various kinds of metadata. We distinguish between (1) static metadata configured by the user (e.g., a TensorFlow model with training data in an S3 bucket), (2) instance metadata (e.g., a deployed instance of the trained model), and (3) runtime metadata (e.g., performance metrics of the model). The static metadata can be specified in a YAML file (or via a graphical UI).

[1]https://aws.amazon.com/sagemaker
[2]https://ibm.com/cloud/machine-learning

Figure 1 illustrates a simple pipeline from use case UC3 to train a TensorFlow model, compress the model using fixed point quantization [13] and export it to CoreML format, and finally deploy the model (to a mobile device) with drift monitoring enabled. Note the `harden:true` and `monitor_drift:true` attributes which cause ModelOps to automatically insert two additional pipeline steps for model hardening and deploying a model drift detector. Hence, the declarative pipeline with three tasks effectively gets transformed into a pipeline with five tasks at instantiation time.

Note the `platform` attribute specifies that the model should be trained on IBM Watson Machine Learning (`wml`). Switching to a different provider or even an on-premise cluster is as simple as changing that attribute (e.g., `aws`, `gcp`). The domain abstractions in the metamodel and plugin system automatically adjust the pipeline accordingly.

```
models:
  - name: my_model_1
    type: tensorflow
    platform: wml
    training_data: s3://mybucket/training_images
pipelines:
  - name: train_deploy_pipeline
    tasks:
      - name: Train Model
        _type: modelops.task.TrainModel
        model: my_model_1
        harden: true
      - name: Compress Model
        _type: modelops.task.CompressModel
        model: my_model_1
        output: CoreML
      - name: Deploy Model
        _type: modelops.task.DeployModel
        model: my_model_1
        monitor_drift: true
```

Listing 1. Pipeline Configuration Sample

B. Pipeline Templating

ModelOps provides a templating approach that allows users to easily bootstrap their configurations based on pluggable pipeline capabilities. We distinguish between pipeline *templates* and pipeline *transformers*, as illustrated in Figure 6. The former are building blocks of common configuration patterns that can be instantiated by the user, whereas the latter are composable pieces of logic that enrich existing configurations with additional features.

In the example in Figure 6, a simple template with three pipeline tasks is defined. Note that, in addition to the pipeline itself, the template also defines configuration entities like models, data stores, and accounts which are required to execute the pipeline. The entities are associated with parameters (e.g., bucket name of the training data store) whose values are provided by the user upon instantiation. The figure also illustrates three sample transformers, along with the structural changes they apply to the pipeline: (1) a transformer that adds a data bias check to the pipeline, (2) a transformer that branches out a separate model version for A/B testing, and (3) a transformer that publishes metadata events for each task to enable detailed model lineage tracking.

Pipeline templates and transformers are easily extensible and can be provided as plugins, for example in a separate Github repository, or as a `pip` Python package. We are currently in the process of releasing the ModelOps framework as

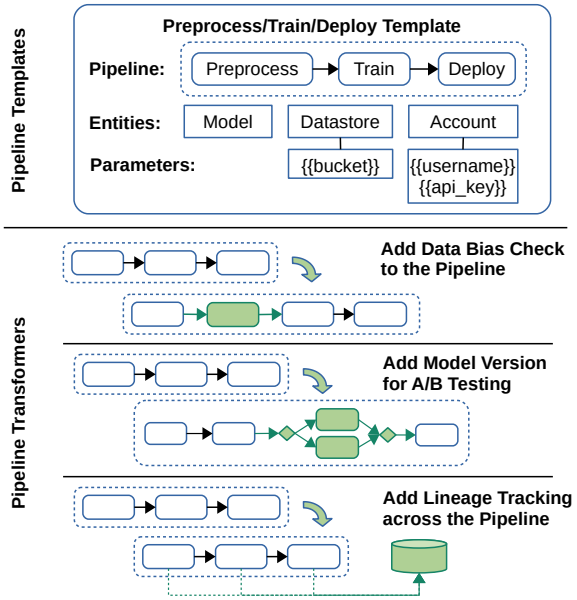

Fig. 6. Parameterizable Pipeline Templates and Transformers

open source, to leverage contributions from the community and build a comprehensive catalog of pluggable and composable pipeline templates.

C. Pipeline Execution

ModelOps pipelines are configured in a generic format that allows us to map it to different target execution platforms. In our current implementation, we have successfully mapped ModelOps pipelines to Apache Airflow[3], OpenWhisk Composer[4], Jenkins[5], as well as Kubernetes/Argo[6]. Figure 7 illustrates the high-level code generation approach. The generic pipeline definition (typically represented in a YAML file, or using a Python based DSL) is combined with a set of task implementations in the task catalog, and ModelOps then generates code for the respective backend system (Airflow DAG, Jenkins pipeline, Composer program, or Argo workflow). This approach provides the flexibility to plug in new execution platforms, and ensures that the catalog of task implementations is fully reusable across these different environments.

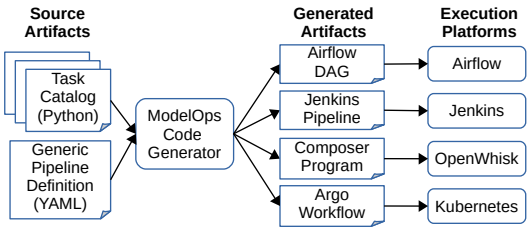

Fig. 7. Pipeline Code Generation and Execution

[3]https://airflow.apache.org
[4]https://github.com/ibm-functions/composer
[5]https://jenkins.io
[6]https://github.com/argoproj/argo

V. Related Work

Integrating DevOps principles and AI is relatively nascent, as discussed by [16], but tentative efforts to scale and automate AI workflows can be seen in industry and academia. Uber has developed Michelangelo [17] for developing, managing, and deploying machine learning models at scale. Industry specific AI platforms are being shaped, e.g., building analytics and recommendation models for video gaming using an agent store to automate experimentation and retraining [18]. Ali et al. [19] developed an ML based sales prediction system for automative CRM supporting model training/re-training, storage, and deployment. Tovar et al. [20] study automatic dependency management for scientific applications on clusters, where a package graph is transformed into a workflow graph, which then gets executed on a target compute cluster. There is also comprehensive related work on automated machine learning (AutoML) [21], with iterative pipelines guided by a search algorithm to find optimal models over large hyperparameter search spaces. Yet, the aforementioned offerings still do not address the complete AI lifecycle with pluggable pipelines and co-evolution of models and applications.

Google introduced the TFX platform which also focuses on pipelines for data processing and model training [22]. TFX uses lower level configuration details, including a proprietary schema for training data as well as a TensorFlow model specification API, whereas ModelOps focuses on domain abstractions for platform-independent pipelines, allowing to easily plug in lifecycle capabilities (bias, robustness, drift, etc) for building and operationalizing AI models. Perhaps the closest effort to our work is Algorithmia's AI Layer product [23]. The AI Layer attempts to automate DevOps for machine learning and deep learning models by providing several capabilities such as pipelining, versioning, and infrastructure optimization. Our framework complements what Algorithmia's AI layer provides by allowing pluggable AI domain abstractions such as bias checking, drift detection, etc. Li et al. [24] discuss challenges associated with building a scalable ML service, including feature computation over global data. Their focus is mainly on real-time serving of large number of models, without considering the integration lifecycle.

Polyzotis et al. [25] have reported on data management challenges in production machine learning pipelines. ModelHub [26] and ModelDB [27] are lifecycle management systems for deep learning models to support efficient storing, querying, and sharing of artifacts, but they do not consider a generalized view on the co-evolution of models and applications. Recent work on trust in AI emphasizes the need for a supplier's declaration of conformity (SDoC) to describe the lineage of AI models along with the safety and performance testing it has undergone [28]. This factsheet-based certification of AI models directly relates to ModelOps, as it provides the underlying events and metadata to track changes across the lifecycle. We are actively working on evolving the concepts of AI factsheets and the ModelOps system requirements derived from them.

The accumulation of technical debt in ML systems was addressed in [29] where different machine learning specific contributing causes of technical debt including models, data, system-design, system configuration, and their mutual interactions are discussed. ModelOps can potentially reduce this debt by DevOps-style standardization of the AI model lifecycle.

VI. Conclusion

Many teams are still struggling to leverage the full potential of AI in their applications, partly due to the investment in skills, tooling, and platforms needed to support AI lifecycles while meeting enterprise governance requirements. AI operations support is critical to narrowing these gaps by allow these teams to more easily incorporate AI technologies in their applications, whether it be systematic versioning of model artifacts, or building reusable pipelines with quality gates that guarantee that deployed models are certified and meet the business ever-changing needs and regulations.

Our work aims to build a foundation for the emerging field of AI lifecycle management. Based on a review and survey of existing use cases in industry and research, we identify the key opportunities and challenges in the field. The ModelOps platform is designed around the concepts of metadata versioning, AI domain abstractions, re-usable patterns, event-based pipelines, and seamless integration of lifecycle capabilities. Our prototype implementation, denoted *ModelOps*, rigorously follows the principles of pluggability, extensibility, consumability and platform independence all in a cloud-native setting. For example, sophisticated algorithms that check and repair vulnerabilities in a model could be incorporated into a model building pipeline with a few lines of code. The AI domain abstractions enable teams to use bias checking, drift detection, or model robustness algorithms without excessive configuration or knowledge of the details of how these algorithms work or where they run.

Our goal is to enable users to continuously evolve and improve their AI models across the lifecycle, systematically reduce and manage the risks of model deployments, and ultimately build more reliable and trusted AI applications. In future work, we plan to onboard more large-scale use cases onto the ModelOps platform, to further validate and evolve the domain concepts and programming model for operationalizing AI across cloud platforms. Moreover, as we collect more data about the structure and characteristics of users' pipeline executions, we are targeting the cloud provider's perspective and investigate techniques for optimized scheduling under quality and resource constraints.

Acknowledgements

We would like to thank Anupama Murthi, Punleuk Oum, Gaodan Fang, and Debashish Saha for their insights and core contributions to the ModelOps architecture and codebase.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[3] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[4] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure devops," in *International Conference on Cloud Engineering (IC2E)*. IEEE, 2016, pp. 202–211.

[5] B. Bhattacharjee, S. Boag, C. Doshi, P. Dube, B. Herta, V. Ishakian *et al.*, "IBM Deep Learning Service," *IBM Journal of Research and Development*, vol. 61, no. 4, pp. 10–1, 2017.

[6] S. Boag, P. Dube, B. Herta, W. Hummer, V. Ishakian, K. R. Jayaram, M. Kalantar, V. Muthusamy, P. Nagpurkar, and F. Rosenberg, "Scalable multi-framework multi-tenant lifecycle management of deep learning training jobs," in *Workshop on ML Systems at NIPS*, 2017.

[7] V. Zantedeschi, M.-I. Nicolae, and A. Rawat, "Efficient defenses against adversarial attacks," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '17. New York, NY, USA: ACM, 2017, pp. 39–49. [Online]. Available: http://doi.acm.org/10.1145/3128572.3140449

[8] IBM Corporation, "IBM AI Fairness 360," http://aif360.mybluemix.net/, 2018.

[9] A. Dhurandhar, P. Chen, R. Luss, C. Tu, P. Ting, K. Shanmugam, and P. Das, "Explanations based on the missing: Towards contrastive explanations with pertinent negatives," *CoRR*, vol. abs/1802.07623, 2018. [Online]. Available: http://arxiv.org/abs/1802.07623

[10] IBM Corporation, "IBM Watson Machine Learning," https://developer.ibm.com/clouddataservices/docs/ibm-watson-machine-learning/, 2018.

[11] K. M. Hazelwood, S. Bird, D. M. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia *et al.*, "Applied machine learning at Facebook: A datacenter infrastructure perspective," 2018, pp. 620–629. [Online]. Available: https://doi.org/10.1109/HPCA.2018.00059

[12] D. Vijaykeerthy, A. Suri, S. Mehta, and P. Kumaraguru, "Hardening deep neural networks via adversarial model cascades," *CoRR*, vol. abs/1802.01448, 2018. [Online]. Available: http://arxiv.org/abs/1802.01448

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv:1510.00149*, 2015.

[14] S. Shaikh, H. Vishwakarma, S. Mehta, K. R. Varshney, K. N. Ramamurthy, and D. Wei, "An end-to-end machine learning pipeline that ensures fairness policies," *CoRR*, vol. abs/1710.06876, 2017. [Online]. Available: http://arxiv.org/abs/1710.06876

[15] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel, "Evaluating the robustness of neural networks: An extreme value theory approach," *arXiv preprint arXiv:1801.10578*, 2018.

[16] Gartner. (2017) Integrate DevOps and Artificial Intelligence to Accelerate IT Solution Delivery and Business Value. https://www.gartner.com/doc/3787770/integrate-devops-artificial-intelligence-accelerate.

[17] J. Hermann and M. Del Balso, "Meet michelangelo: Ubers machine learning platform," *URL https://eng.uber.com/michelangelo*, 2017.

[18] J. F. Kolen, M. Sardari, M. Mattar, N. Peterson, and M. Wu, "Horizontal scaling with a framework for providing ai solutions within a game company," 2018.

[19] M. Ali and Y. Lee, "CRM Sales Prediction Using Continuous Time-Evolving Classification," in *AAAI Conf.*, 2018.

[20] B. Tovar, N. Hazekamp, N. Kremer-Herman, and D. Thain, "Automatic dependency management for scientific applications on clusters," in *International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 41–49.

[21] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2962–2970.

[22] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2017, pp. 1387–1395.

[23] Algorithmia, "Enterprise AI Layer," https://algorithmia.com/enterprise, 2018.

[24] L. E. Li, E. Chen, J. Hermann, P. Zhang, and L. Wang, "Scaling machine learning as a service," in *Int. Conf. on Predictive Applications and APIs*, vol. 67, 2017, pp. 14–29.

[25] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data management challenges in production machine learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 1723–1726.

[26] H. Miao, A. Li, L. S. Davis, and A. Deshpande, "Towards unified data and lifecycle management for deep learning," in *Int. Conf. on Data Engineering (ICDE'17)*, April 2017, pp. 571–582.

[27] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "ModelDB: A System for Machine Learning Model Management," in *Workshop on Human-In-the-Loop Data Analytics*, 2016, p. 14.

[28] M. Hind, S. Mehta, A. Mojsilovic, R. Nair, K. N. Ramamurthy, A. Olteanu, and K. R. Varshney, "Increasing trust in ai services through supplier's declarations of conformity," *arXiv:1808.07261*, 2018.

[29] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *28th International Conference on Neural Information Processing Systems*, ser. NIPS'15, 2015, pp. 2503–2511.