AI-BASED SYSTEM FOR ADVANCED ANALYSIS OF PIPELINE CRACK-DETECTION DATA IN FLUID TRANSPORTATION

PhD thesis – Abstract

submitted for obtaining the Scientific Title of PhD in Engineering from
Politehnica University Timișoara, Romania
in the Field of Systems Engineering

by Eng. Adrian-Ioan ARGEŞANU
PhD Supervisor PhD Eng. Prof. Gheorghe-Daniel ANDREESCU

December 2023

Pipelines are the primary means to safely and efficiently transport oil or gas at high pressures across long distances. A main focus is set on the safety and availability of the pipeline system. From the integrity perspective pipelines can be treated as pressure vessels. Any anomaly or flaw must be detected and identified before it has a detrimental effect on the integrity of a given line or system [3]. Therefore, the integrity of the pipelines needs to be monitored frequently by pipeline operators. With more than 90% of the pipelines buried underground [1], locating potential flaws is a major challenge.

There is a multitude of features and defects in pipelines that can lead to their failure over time. They can originate from various sources such as pipeline operating parameters, ambient conditions or even the pipe manufacturing process. Generally, pipeline defects are classified into the following major categories: metal loss, cracks or crack-like defects, laminations and other mid-wall defects, and geometric anomalies. Whilst metal loss is by far the most widespread phenomenon, cracking often is the more worrying threat to pipeline safety due to its potential for sudden failure.

As indicated by its name, stress corrosion cracking (SCC) is known to develop under simultaneous presence of tensile stress and corrosion attack. Theoretically, it can initiate at any point where the local hoop stress surpasses the actual local resistance of the material. External SCC typically develops as very fine hairline cracks not even visible to the human eye. Their orientation is usually longitudinal, and they occur in so-called “colonies”. With all these fine cracks oriented in the same direction and with some of them lengthening and deepening over time, the cracks within a colony may eventually join together. Since SCC initially develops slowly, crack colonies can exist in pipelines for many years without having an impact on the pipeline operation. If the cracks coalesce and grow to a critical size, however, the pipeline will either leak or rupture. Due to the fact that reliable prediction of SCC occurrence is not possible, pipeline operators resort to regular inspection surveys.

Conventional pipeline integrity testing methods are performed 'in-line' using intelligent pipeline inspection gadget (PIGs). Smart PIGs include electronics and sensors that collect various forms of data during the trip through the pipeline. Given the structural nature of pipeline cracks, an ultrasonic based inspection technique leveraging piezoelectric transducers is the best detection method to use [4]. Breaks in the homogeneity of the metal grid (i.e., defects such as cracks, such as the samples depicted in Figure 1) will result in reflections of the sound wave. Via the direct piezoelectric effect, these reflected waves are translated to a current in the wire. This current is measured, processed and analyzed. The signal’s characteristics and its time of receipt, when combined with that of other sensors, provide information about the feature’s

geometry and position in the pipe.

UltraScan CDP (Process & Pipeline Services' ultrasonic in-line inspection (ILI) tool, depicted in Figure 1) was designed to detect and size longitudinally oriented cracks and crack colonies as short as 25 mm and as shallow as 1 mm in pipelines transporting liquid mediums at inspection speeds of up to 5 m/s [2]. Depending on the diameter of the to-be-inspected pipeline, up to several hundred piezoelectric transducers are used to scan the condition of the pipe wall.

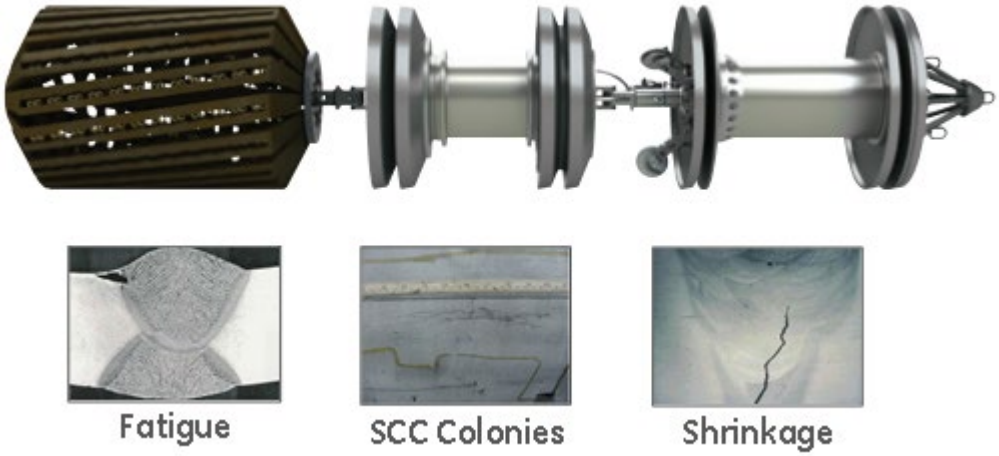

Figure 1. UltraScan CDP [2] (top) and sample crack categories [11] (bottom)

Due to the large amount of signal data recorded during runs of potentially more than one hundred kilometers, one of the major performance bottlenecks of early ILI tools was data storage. Several approaches have been made to overcome this issue using different types of data compression. In particular, UltraScan CDP resorts to the ALOK (Amplitude Time Locus Curves) compression method, which provides lossy data compression but very high compression ratios. In parallel to compression, the recorded data is scanned for events of interest in real-time, allowing further reduction of required storage space by discarding any sequence of signals which does not show any defect-like patterns. The result is stored in flat binary files.

The compression strategy employed by Process & Pipeline Services' crack detection tool results in a very sparse representation of the pipe wall being stored and made available for further analysis. Aggregation of all compressed values recorded by one sensor during one inspection cycle to an A-Scan results in a 2D representation of amplitudes vs time-of-flight. Further aggregation of all A-Scans recorded by a particular sensor to a B-Scan results in a 3D representation of amplitudes vs time-of-flight vs distance. The highest-level aggregation involves overlaying all B-Scans recorded by all sensors into a 4D view, which is flattened on the azimuth axis to a 3D C-Scan representation. The C-Scan view depicts the entire pipe circumference. Sample C- and B-Scan views are showcased in Figure 2; the amplitudes are coded in false color.

Post analysis, a report is forwarded to the pipeline operator, which proceeds to the excavation of pipeline sections in order to repair detected defects. Because of the huge financial cost of excavations, data analysis with minimal errors is of the utmost importance. Whilst the recorded reflections contain information regarding the nature and sizing of the features encountered, defect detection, classification, and accurate sizing using guided ultrasonic waves is still a major challenge under investigation. Due to the complexity of the wave propagation

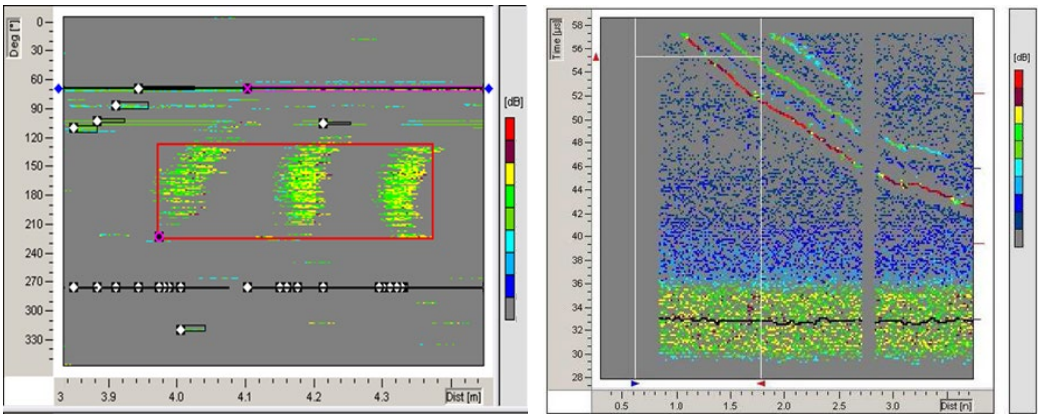

Figure 2. Sample C-Scan and B-Scan of UT data

characteristics, significant manual effort from experienced subject-matter-experts (SME) is required during the analysis process.

To reduce this effort, algorithms are employed wherever possible. Algorithm results are rarely directly published in the customer-facing report; these are typically manually QA'd by the SMEs. For analysis of data recorded by UltraScan CDP, one of the main points of interest is the automation designed to detect and highlight areas of potential crack-like defects. Due to safety considerations, this detection algorithm is tuned to not miss any significant defect, thus making it highly sensitive to changes in pipeline characteristics, such as manufacturing details, medium parameters and pipeline cleanliness. Coupled with the challenges arising from the sparseness of the recorded data, this translates into a high number of areas of interest – also called regions of interest (ROIs) – created around non-defect reflectors such as rough pipe wall surface, wax and other non-metallic deposits, impurities in the medium, electronical noise, sensor cross-talk, etc.

Figure 3 depicts a summary of the assessment steps. After the data is downloaded from the tool it is subject to automated algorithms, with the results of the automations representing the input for manual analysis (referred to as "Data Analysis" in the figure). With "Data Analysis" concluded, the "Client Deliverables" – reports, listings, statistics – are compiled and sent out to the pipeline operator.

The more ROIs are created around non-defect reflectors, the more time is required for SME analysis and thus the less efficient the process becomes. To enable SMEs to focus on the "high-skill" decisions, the number of non-defect ROIs must be reduced. Since safety considerations prohibit more aggressive tuning of the automation, the focus is shifted on pre-filtering of its results before manual SME assessment.

The main objective of this work was to develop and roll out an intelligent bespoke decision-making tool to aid the SMEs in distinguishing typical defect patterns from non-defect reflectors in crack detection ILI data as recorded by Process & Pipeline Services' UltraScan CDP inspection tool.

With training and validation of such a system requiring high-throughput batch-access to the highly compressed 4D binary structures, first **a NoSQL data storage solution** was engineered. NoSQL databases, also known as "Not only SQL", "non-relational" and "non-SQL", have existed since the late 1960s [7], but have only achieved general acceptance in the early 21st century, serving big data and real-time web applications [8].

These data stores represent an alternative to traditional relational databases and do not require a fixed schema. This in turn means that these can store structured, semi-structured, unstructured and polymorphic data. This aspect makes NoSQL stores particularly efficient for

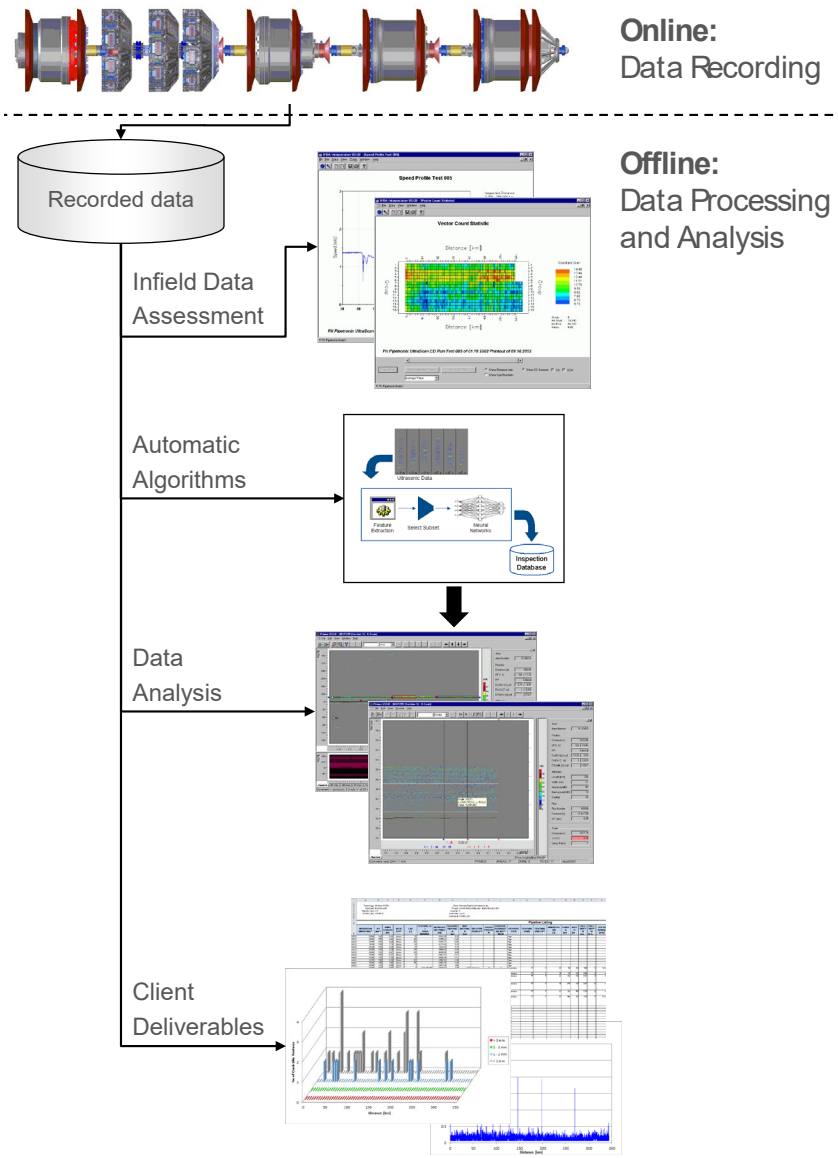

Figure 3. Data flow [11]

sparse data.

NoSQL stores have been designed to serve vast volumes of data, trading the traditional scale-up (vertically) with a scale-out (horizontally) approach to satisfy latency requirements with increasing load.

I performed a comprehensive comparative study of existing NoSQL technologies in the context of Process & Pipeline Services' data storage requirements – one-off ingestion, batch processing, and interactive assessment. HBase was identified as the best-fit candidate for UltraScan CDP's 4D data. HBase is a wide-column database based on Google's BigTable [9]. Designed for storing extremely large quantities of rows in a table-like structure made up of extremely large numbers of columns, HBase claims to offer fast and random access to the hosted data, in addition to various other features and benefits:

- A flexible data model that supports data that is highly-variable in its structure,
- Strictly consistent reads and writes with support for real-time queries,
- Scalable, with automatic and configurable sharding of database tables.

For HBase, a custom-developed soft-schema was introduced – "one A-Scan per row" (Figure 4), storing, as the name implies, the data recorded during one inspection shot by one sensor per row. Abiding by the tall but narrow soft-schema design, "one A-Scan per row" exhibited the best compromise for range-scans of various lengths of UltraScan CDP's sparse inspection data.

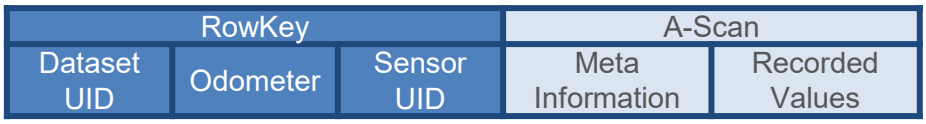

Figure 4. HBase schema: one A-Scan per row – detailed view

To address HBase's main data ingestion drawback, I developed and introduced a software solution to accelerate bulk data ingress into range-scan optimized HBase clusters. Preventing Region and Region Server congestion, Pre-Shard and Shard-Allocation (Figure 5) ensured the HBase table remained optimized for data egress even before the first write request was issued, at the cost of minimal compute overhead, without interfering with the existing data or row-key design.

In the first step, called Pre-Shard, all necessary table splits are provisioned for the imminent ingestion. The Pre-Shard utility evaluates the to-be-ingested dataset and, considering the row-key structure, derives the key range it will occupy in the targeted HBase table. Assuming a RegionSplitPolicy is already enforced on the table for optimal partitioning, Pre-Shard leverages the configured policy and its parameters to compute the split points for the key range. The user can also specify a different RegionSplitPolicy at runtime – e.g., when the partitioning of the table is too coarse for the inbound dataset, to customize Region and Region Server load for each ingested dataset.

In vanilla HBase, the Balancer would eventually detect the Regions provisioned by Pre-Shard, and with all residing on a single node, pick a subset of these for relocation to other nodes.

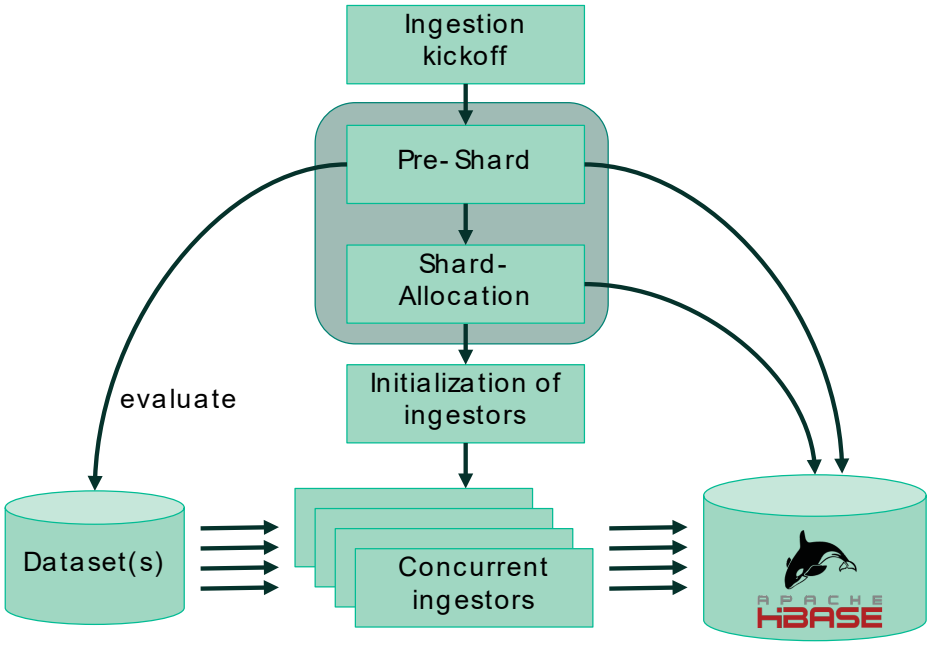

Figure 5. Integration of Pre-Shard and Shard-Allocation

Since this automation does not consider key range vicinity and its effects on bulk-ingestion or range-scanning, Region Server hotspotting is likely to occur during the actual ingestion or subsequent read workloads. Addressing this aspect, a Shard-Allocation component was engineered. The utility queries all newly provisioned Regions and active Region Servers from HMaster and assigns the shards to nodes in a round-robin fashion, such that no two shards with neighboring key ranges are located on the same node. With step two completed, the write workload can be resumed as normal.

Direct comparison of ingestion performance in the test setup showed a 76% decrease in number of block-and-wait events when Pre-Shard and Shard-Allocation were enabled (Figure 6). This equated to 4.2x more records ingested on average between block-and-wait events and a 47% average reduction of ingestion duration. [12] Pre-Shard and Shard-Allocation are applicable to all HBase instances storing timeseries-like data.

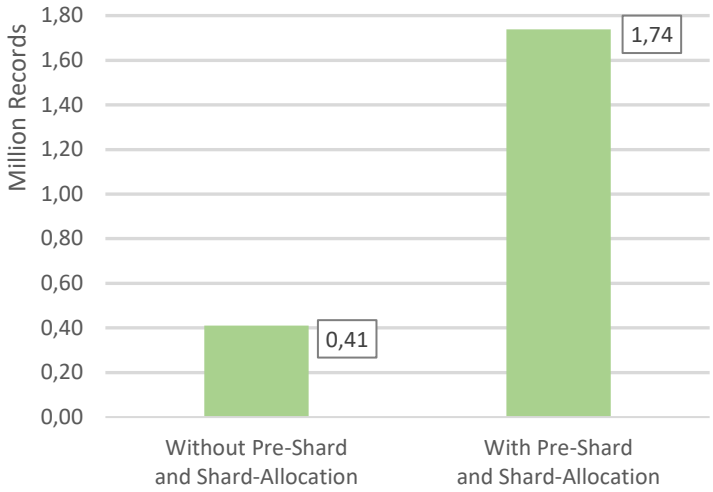

Figure 6. Average nr. of records ingested between block-and-wait events

With Process & Pipeline Services' historical data preloaded into the NoSQL store, I focused on algorithm prototyping and productionization efforts, introducing a bespoke **platform to manage the end-to-end lifecycle of machine learning (ML) models**. Offering native offline scoring capabilities, the proposed platform complements the likes of Bighead and Michelangelo. [13]

The platform is built in Python and relies on open-source technologies like Docker, Keras, and more. Primarily designed – but not locked down – to run in Amazon Web Services (AWS), the individual components are offered in the form of Docker containers. Each Docker container, referred to as a service, encapsulates a single functionality, typically wrapped in a HTTP server. Thanks to the containerized approach, services can be spun up on local hardware as well as virtualized setups. Services can be aggregated on a per-need basis to address the changing challenges of different machine learning problems. A multi-service setup is referred to as an environment; environments are orchestrated via Docker Swarm.

From an architectural standpoint we distinguish between the following major layers, as depicted in Figure 7: Orchestration, Model Serving, Data Access and Versioning.

The Orchestration layer holds three main components: the Orchestrator, a client facing web service – called Prediction Web Service, and the Orchestration Storage.

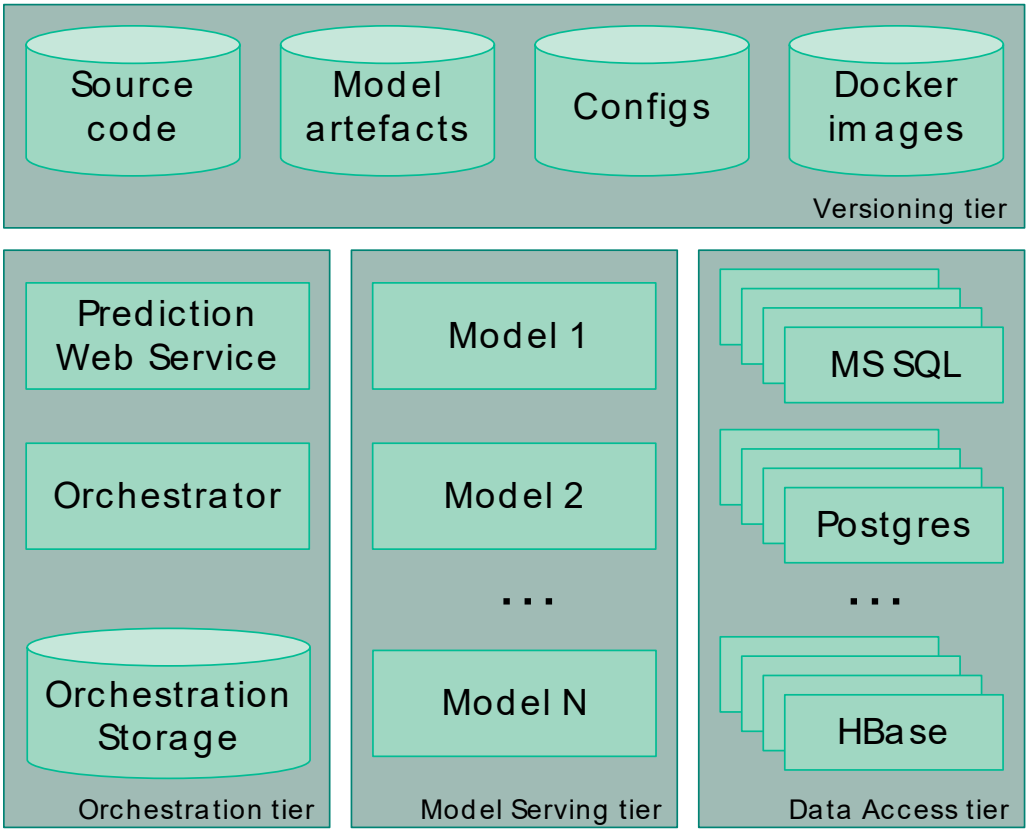

Figure 7. System architecture

The Orchestration Storage is the central information store of the platform and only exposed to the Orchestration tier; one Orchestration Storage is deployed per environment. It is used for configuration, synchronization, checkpointing and recordkeeping. All model predict-time configurations (e.g., service URL, size of prediction batch, service timeout), prediction requests (cataloging information as well as all prediction candidates), progress messages, error notifications, and (prediction) outcomes are persisted in dedicated structures in this storage.

The Prediction Web Service hosts the collection of endpoints the client interfaces with when issuing prediction requests and tracking their progress.

Acting as the central orchestration unit for the environment, the Orchestrator is also in charge of monitoring and management of infrastructure and services, which it facilitates via an integrated frontend. Views are available for various levels of monitoring, debugging and data/model/evaluation visualization and include: Celery Flower, Portainer, Kibana, “Deployed Models” dashboard, “Last Prediction Requests” view, Redash and pgweb.

The Model Serving tier hosts all deployed models. Each model service contains a single model wrapped in an HTTP server. Through Docker image inheritance, all model images share the same framework, API and core functionality – all shared components are available in the so-called “base image”. The base image abstracts away from the model use-case and type of machine learning framework used. Inheriting images can inject their custom dependencies and runtime for maximum flexibility.

The base image provides scaffolding for model (re-)training and prediction serving. Hooks for pre-predict checks (any candidate failing these will not be predicted on), and post-predict safeguards (to capture the cases in which the inference result is on best endeavor and should be interpreted with caution by the requestor) are also provided.

The Data Access tier facilitates data ingestion into the platform. Individual connectors – called DACs – can be coded in different languages and configured depending on the storage medium that holds the input data. APIs vary according to the use-case, but always feature an endpoint for health checking.

All elements of the Versioning tier are available cross-environment and are deployed as singletons. All development efforts, service deployment and orchestration tasks rely on the availability of this tier.

Versioning is applied on multiple levels for all modules, ranging from source code, over model parameters and artefacts, to Docker images. These are stored in a variety of systems, including: Git, Docker Registry and AWS S3. I employ a universally unique identifier scheme to catalogue and model the relationships of all services, models, model parameters and artefacts.

To automate the majority of the model development and deployment steps, a dedicated command line interface (CLI) with customizable contexts has been implemented. The CLI is used for development, building of Docker images, testing of code and images, training and validation of models, deployment via Jenkins Continuous Deployment pipelines, and lifecycle management. CLI commands are available anywhere inside the checked-out copy of the source code.

The CLI facilitates but does not impose regular integration and testing efforts. To ensure the stability of the codebase across commits, a custom CI/CD setup is employed. I configured Jenkins pipelines, to build, test and deploy code, images and services. The Jenkins pipelines leverage CLI functionality wherever possible – e.g., via the `build`, `test`, `up` hooks, and in turn the CLI leverages the Jenkins deploy pipeline for remote rollout of services [20].

To maximize the uptime of the platform, ensure it operates efficiently and track ML model performance, I engineered a multi-tiered monitoring solution. When compared to traditional software systems, ML systems exhibit all the challenges of traditional code, plus an array of machine learning-specific considerations [17]. Unlike in traditional software systems, an ML system's behavior is governed not just by rules specified in the code, but also by model behavior learned from data. The novelty of my approach lies in how the individual components are selected, layered, integrated, and presented to various levels of stakeholders [19]. The solution screens, captures and displays:

- Availability of hardware and its usage (HW uptime and utilization) as part of system monitoring via metric scrapers, Prometheus and Grafana,
- Availability of individual services and their status (service uptime and health) as part of system monitoring via metric scrapers, Prometheus and Grafana,
- Performance of machine learning models (input and safeguard monitoring, prediction

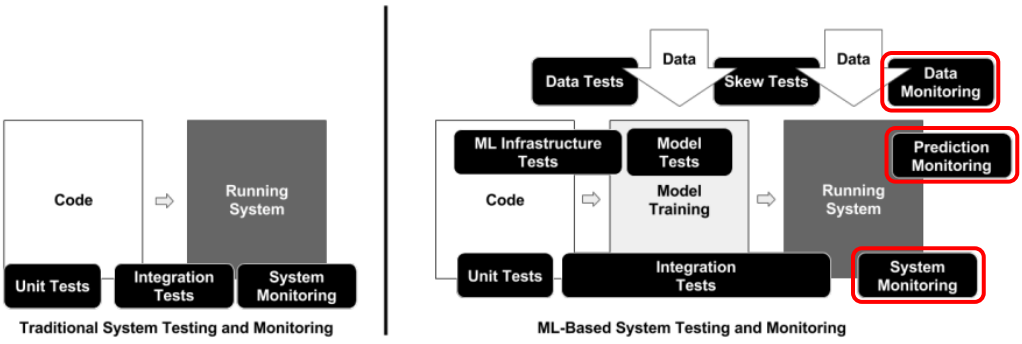

Figure 8. Monitoring of traditional code (left) versus ML systems (right) [15]

outcomes, feedback loops) as a combination of data and prediction monitoring via the Orchestration Storage, Redash and pgweb,

- Event logs from all models and services (auditability and debugging) as a combination of all three monitoring areas highlighted in Figure 8 via the ELK (Elasticsearch, Logstash, Kibana) stack.

The monitoring solution thus offers three dashboards, each catering to a specific level of detail and thus audience. With Grafana as the high-level system view, stakeholders and maintenance personnel have a one-stop dashboard for the health of the application. Leveraging all historic prediction data, Redash can be used to display high-level as well as detailed insights to stakeholders and expert users alike, enabling them to track the behavior of each ML model versus its performance targets. Finally, the Kibana dashboard allows temporal correlation of all events recorded throughout the system, greatly improving failure analysis and debugging activities for development personnel, whilst eliminating the need to manually aggregate information from different sources.

The presented platform aims to make the machine learning process seamless, scalable and consistent. As noted on Process & Pipeline Services' use-case:

- Capturing of the data provenance records for all experiments and versioning all trained models ensured reproducibility and auditability was attained during all stages of the model's lifecycle,
- The system offered close to zero effort deployment in production: the data pipelines of the deployed model were identical to the ones used during training, the dockerized approach ensured the production environment characteristics were indistinguishable from the ones used during development, versioning ensured all trained models and auxiliary services were readily available for deployment,
- During operation, the platform has proven scalable to arbitrarily large data sets – deployed on an AWS g3.4xlarge EC2, the model served batch predictions on several million data points per dataset, for datasets exceeding several TB in uncompressed storage, utilizing up to 64 Data Access connectors in parallel, without stability or memory exhaustion issues,
- For model monitoring I leveraged the Orchestration Storage and the frontend of the Orchestrator. Graphs and statistics were set up for automation percentages (number of published predictions versus total candidates), score distributions, input filters and scoring safeguards. [13]

Leveraging the HBase NoSQL store and the platform for end-to-end lifecycle management of ML applications, I built a **novel solution to support the analysis of the multi-dimensional signal data recorded by the UltraScan CDP crack detection tool through identification and classification of non-injurious signal patterns**. The application was trained on 100+ inspections conducted by the UltraScan CDP ILI tool, aggregating the recorded 4D data with inspection-level meta-information for a combined available set of over 5 million ROIs.

The resulting Convolutional Neural Network (CNN) ensemble model proved capable of classifying ROIs with an extremely high degree of fidelity. Figure 9 shows the score distribution plots for 5 sample inspections. The presented plots are representative for the entire set. Predictions for injurious ROIs are depicted in blue, for non-injurious in magenta. The left-most occurrence (with the lowest prediction scores) of injurious ROI predictions is highlighted with a blue arrow in each plot. Even though the two distributions exhibit quite some overlap in the upper score area, the results indicate that the model allows the isolation of non-injurious samples for automation.

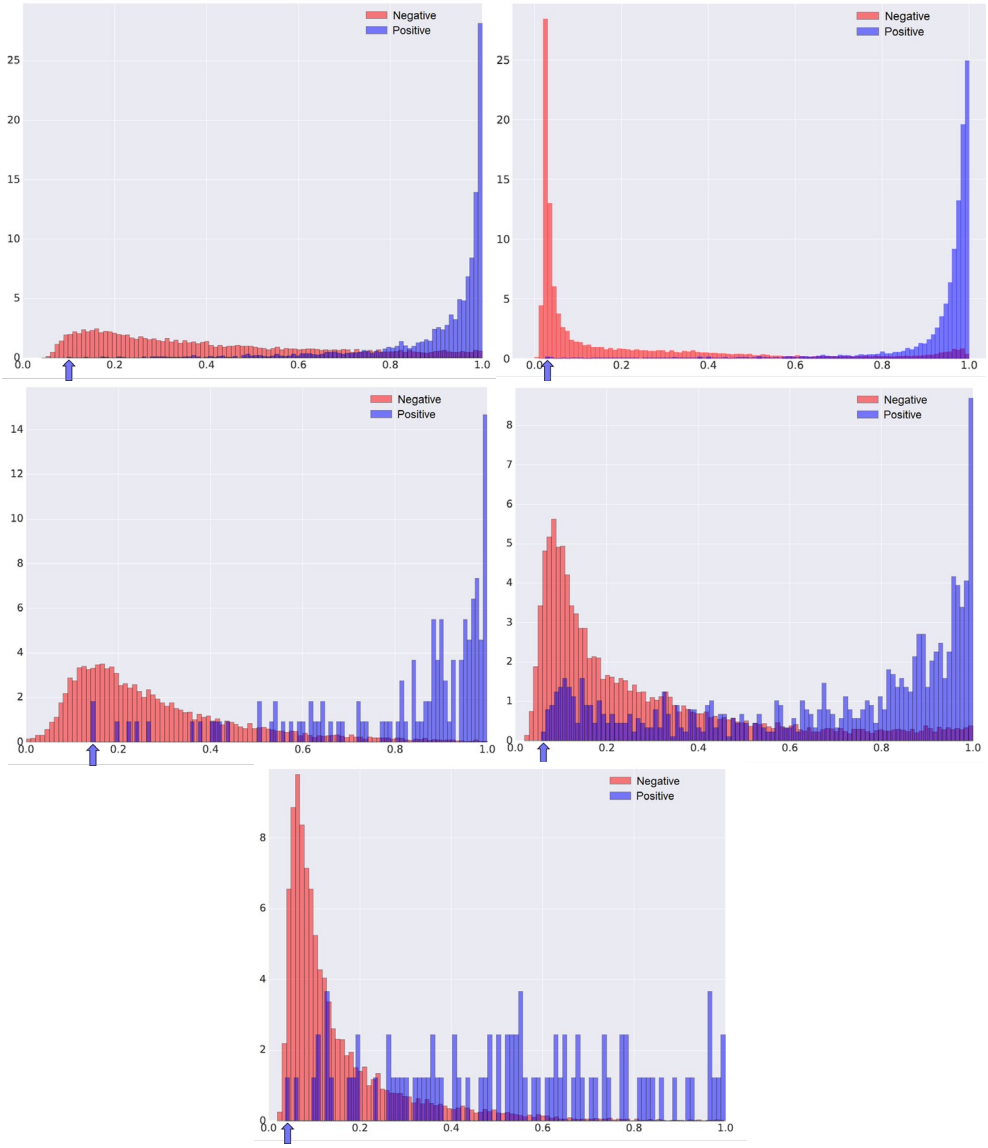

Figure 9. Prediction score distribution per inspection

Through automatic classification of ROIs as non-injurious, I achieved a double-digit percentage reduction in the overall required analysis time for UltraScan CDP datasets. The production model was measured to automate 3.5x as many areas from manual assessment per time unit compared to the peak manual assessment rate of SMEs, thus justifying its rollout into operations. Removal of "routine" work allowed SMEs to focus on "high skill" / "hard" decisions.

For production rollout, the model was fitted with a variety of safeguards to ensure data-/concept-drift and overall degrading model performance was detected before it had a detrimental impact. Rollback procedures were put in place for disaster recovery. The Redash dashboard (Figure 10) was leveraged to visually verify model performance and enable the SME feedback loop. The results depicted in Figure 10 are for one dataset. The top half of the figure depicts the automation percentage per event (left hand side) as well as the inference score distribution (right hand side). Qualified personnel can evaluate this information for abnormal automation rates – too high or too low, as well as population segregation.

The bottom half of Figure 10 presents a summary of the pre-predict filters, followed by

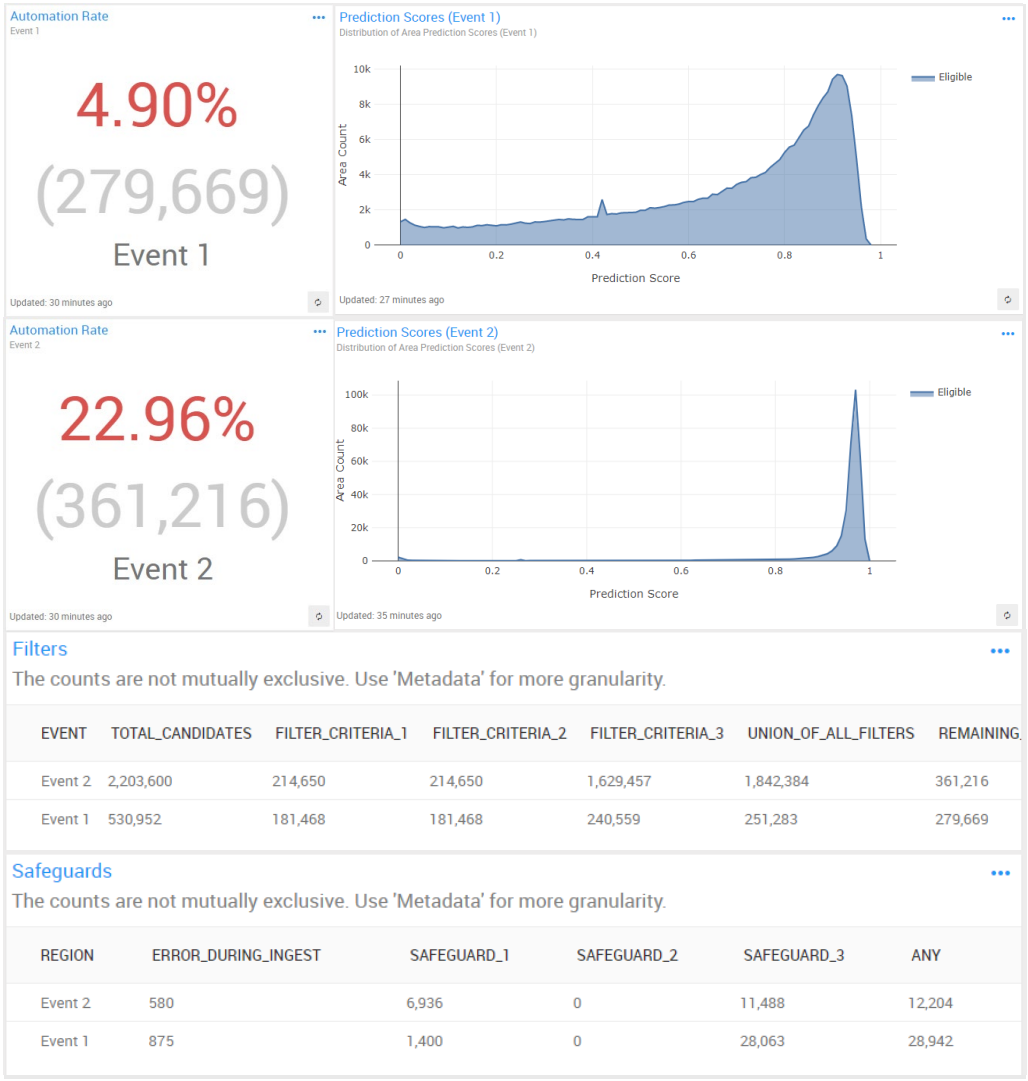

Figure 10. Redash dashboard

a summary of the post-predict safeguards. Any failing pre-predict check is immediately noted in the protocol associated with the prediction candidate and the candidate is disqualified from inference, mitigating safety-outliers. The post-predict safeguards are executed after the model has predicted on the candidate. The main goal is to notify the requester if any of the predictions need to be treated as "best-endeavor". In such scenarios, the outcome of the prediction might still be relevant for augmentation or QA scenarios, but is disqualified for safety-critical automation applications. Any post-predict failure is noted in the protocol of the prediction candidate. The client is responsible for acting on safeguard hits. The depicted model applies the same pre-prediction filters and post-prediction safeguards for both events.

Summarizing the personal contributions of this thesis, these include but are not limited to:

- A comparative study of existing NoSQL technologies in the context of Process & Pipeline Services' data storage requirements,
- A bespoke NoSQL storage solution for Process & Pipeline Services' UltraScan CDP 4D data,
- An accelerated ingestion mechanism applicable to all HBase instances serving range-scan use-cases,

- A newly developed platform to manage the entire lifecycle of ML models with the potential to revolutionize the way ML models are trained and productionized. The platform focuses on automation, repeatability, reproducibility, and observability – aspects which are all discussed in this work,
- A pioneering original solution for the automated analysis of pipeline crack-detection data as recorded by UltraScan CDP. The complex ML model engineered atop the introduced components and systems yields significant automation rates in a safety critical environment, allowing human experts to focus their efforts on where they matter most.

The approaches presented herein are original contributions. As of writing this work:

- No comparable tooling existed to aid the SMEs in distinguishing typical defect patterns from non-defect reflectors for UltraScan CDP ILI data,
- No prior instances or discussions of an equivalent platform to manage the end-to-end lifecycle of batch-prediction machine learning models for in-line inspection data are available in existing literature to the best of our knowledge,
- No analogous NoSQL solution for storage of in-line inspection data or NoSQL ingestion acceleration has been explored in literature to the best of our understanding.

BIBLIOGRAPHY

[1] V. K. Varma, R. W. Tucker, Jr., A. P. Albright, "EMAT-Based Inspection of Natural Gas Pipelines for Stress Corrosion Cracks," Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA, 2004.

[2] "Crack capabilities overview." Process & Pipeline Services. Accessed April 2021. [Online]. Available: https://www.bakerhughes.com/sites/bakerhughes/files/2020-07/19004_BH_PPS_ILI_US_BRO_1912%20%28CRACK%20CAPAB%29.pdf

[3] A. Barbian, M. Beller, "In-Line Inspection of High Pressure Transmission Pipelines: State-of-the-Art and Future Trends," *18th World Conference on Nondestructive Testing*, April 2012, Durban, South Africa.

[4] W. Krieg, "A Novel EMAT Crack Detection and Coating Disbondment (RoCD2) ILI Technology," *Pipeline Technology Conference*, 2007, Hannover, Germany.

[5] N. V. Chawla, "Data Mining for Imbalanced Datasets: An Overview," in Maimon O., Rokach L. (eds), *Data Mining and Knowledge Discovery Handbook*, Springer, Boston, MA, 2009.

[6] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, A. Geiger, "Sparsity Invariant CNNs," *2017 International Conference on 3D Vision (3DV)*, Qingdao, China, 2017, pp. 11-20, doi: 10.1109/3DV.2017.00012.

[7] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," in *Computer*, vol. 43, no. 2, pp. 12-14, Feb. 2010, doi: 10.1109/MC.2010.58.

[8] "RDBMS dominate the database market, but NoSQL systems are catching up." DB-Engines.com. Accessed 25 November 2019. [Online]. Available: https://db-engines.com/en/blog_post/23

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," Google Inc., 2006. [Online]. Available: https://research.google.com/archive/bigtable-osdi06.pdf

[10] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, D. Dennison, "Hidden Technical Debt in Machine Learning Systems," *NIPS*, 2015, pp. 2494-2502.

[11] "UT Inline-Inspection Overview," Process & Pipeline Services, unpublished.

[12] A.-I. Argesanu, G.-D. Andreescu, "Accelerating Data Ingress for Range-Scan Optimized HBase Instances," *2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, Romania, 2021, pp. 341-344, doi: 10.1109/SACI51354.2021.9465633.

[13] A.-I. Argesanu, G. D. Andreescu, "A Platform to Manage the End-to-End Lifecycle of Batch-Prediction Machine Learning Models," *2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, Romania, 2021, pp. 000329-000334, doi: 10.1109/SACI51354.2021.9465588.

[14] L. Bellatreche, G. Chernishev, A. Corral, S. Ouchani, J. Vain, "Advances in Model and Data Engineering in the Digitalization Era," *MEDI 2021 International Workshops: DETECT, SIAS, CSMML, BIOC, HEDA*, Tallinn, Estonia, June 2021, doi: 10.1007/978-3-030-87657-9.

[15] E. Breck, S. Cai, E. Nielsen, M. Salib, D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," *2017 IEEE International Conference on Big Data (Big Data)*, Boston, MA, USA, 2017, pp. 1123-1132, doi: 10.1109/BigData.2017.8258038.

[16] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *3rd International Conference on Learning Representations (ICLR) 2015*, May 2015.

[17] E. d. S. Nascimento, I. Ahmed, E. Oliveira, M. P. Palheta, I. Steinmacher, T. Conte, "Understanding Development Process of Machine Learning Systems: Challenges and Solutions," *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Porto de Galinhas, Brazil, 2019, pp. 1-6, doi: 10.1109/ESEM.2019.8870157.

[18] S. Shankar, R. Garcia, J.M. Hellerstein, A.G. Parameswaran, "Operationalizing machine learning: An interview study," arXiv preprint arXiv:2209.09125, 2022.

[19] A.-I. Argesanu, G.-D. Andreescu, "From Data to Decisions: The Importance of Monitoring ML Systems in Industrial Settings," *Acta Technica Napocensis - Series: Applied Mathematics, Mechanics, and Engineering*, vol. 66, no. 3, 2023. [Online]. Available: https://atna-mam.utcluj.ro/index.php/Acta/article/view/2188

[20] A.-I. Argesanu, G.-D. Andreescu, "Streamlining Machine Learning Workflows in Industrial Applications with CLI's and CI/CD Pipelines," *Acta Technica Napocensis - Series: Applied Mathematics, Mechanics, and Engineering*, vol. 66, no. 3, 2023. [Online]. Available: https://atna-mam.utcluj.ro/index.php/Acta/article/view/2189