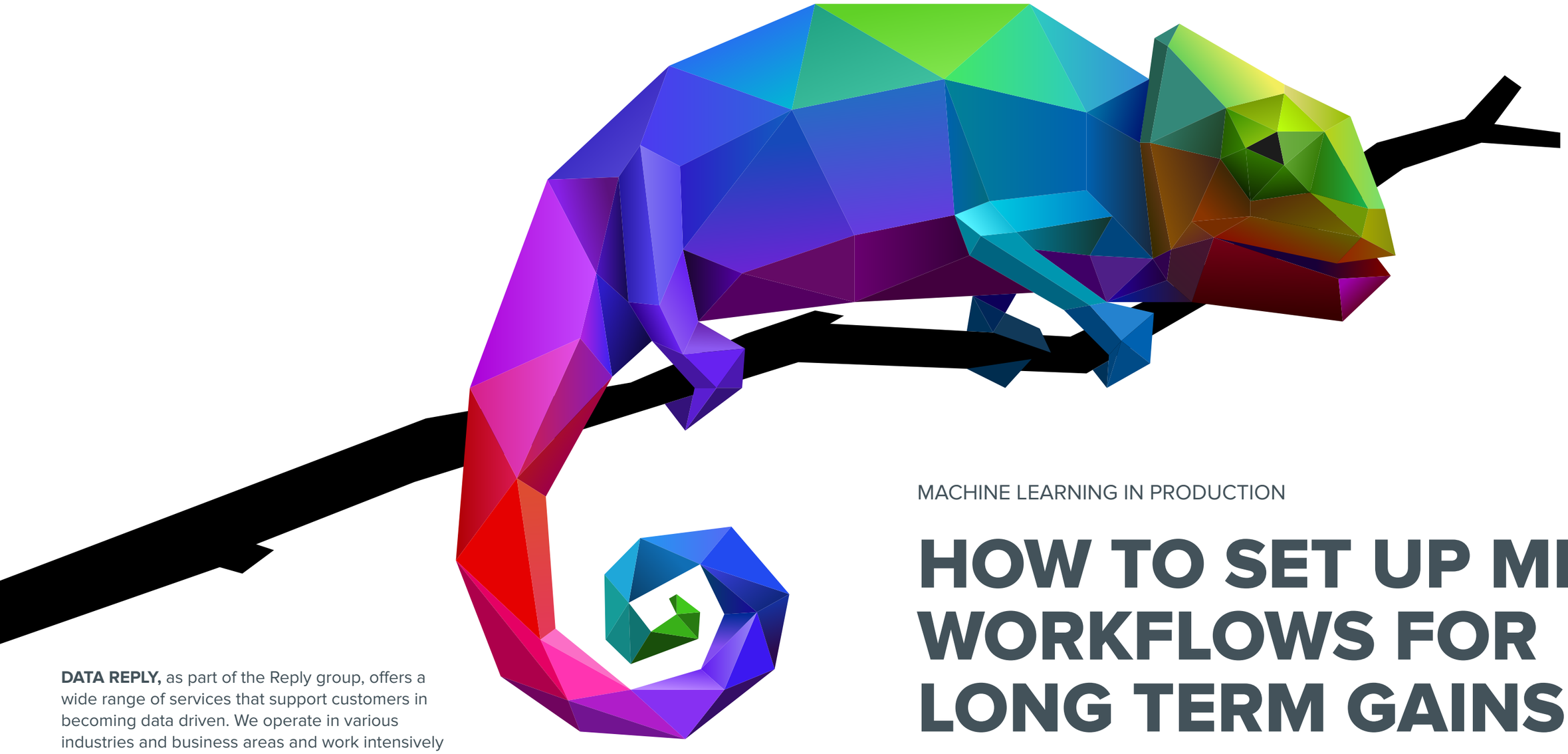


DATA REPLY, as part of the Reply group, offers a wide range of services that support customers in becoming data driven. We operate in various industries and business areas and work intensively with our customers so that they can achieve meaningful results through the effective use of data. Data Reply offers many years of experience in transformation projects to achieve “data-driven companies”. We focus on the development of data platforms, machine learning solutions and streaming applications - automated, efficient, and scalable - without making any compromises in IT security.

www.data.reply.com
infodatareply@reply.de

MACHINE LEARNING IN PRODUCTION

HOW TO SET UP ML WORKFLOWS FOR LONG TERM GAINS

From neural machine translation models which reduce translation times of texts to applying natural language processing (NLP) algorithms to customer data in order to personalize offers: The benefits of software products using Machine Learning (ML) are immense. When developing such applications, the chosen development approach has a huge impact on the success.

THE KEYS

Integrating long-standing knowledge from software engineering practices into the machine learning development lifecycle and the effective use of modern technologies and concepts such as:

- Infrastructure as Code
- CI/CD over several environments & branch management
- Containerization
- Automation and Scalability

The approach described in the following aims to translate recent advances in ML algorithms into software products which are easily accessible, maintainable and upgradeable. At the end, the adaptability of the approach is illustrated with two use cases:

1. Neural Machine Translation API with domain adaption for the BMW Group leading to an efficient, fast, low-cost translation service on AWS shared across the whole enterprise
2. Personalized offers to customers in retail, based on actual customer needs identified through Natural Language Processing (NLP)

HOW TO SET UP ML WORKFLOWS FOR LONG TERM GAINS

CONTENT

TRADITIONAL ML DEVELOPMENT PATTERNS

Over many years Data Reply has witnessed a wide usage of what is called “traditional machine learning (ML) development practices”. These techniques were sufficient during initial investigation of the industrial suitability of ML techniques, but often contributed heavily to the Proof of Concept (PoC) trap, where projects stay in PoC stage without managing to mature to production grade ones. Hence, they are now being superseded by practices which are better suited for developing production software products.

LOCAL DATA SAMPLES AND RESULT STORAGE

The most common way to work with data and analyse results is using locally cached samples. While this is initially sufficient for small-scale problems it rapidly breaks down as the demand for memory and storage increases. More importantly, it does not lend itself to straight-forward generalization of production deployments where data storage and computational environments exist remotely.

SCRIPTS & UNSTRUCTURED CODING

The ability to rapidly explore data and try out new hypotheses is a crucial component of PoC work with machine learning. In fact, the innovative nature of the field was one of the key features making it appealing to the industry. The use of scripts and unstructured coding environments has however several disadvantages which have become increasingly apparent as the field matured. The purely exploratory mindset encourages a tendency to disregard structured coding practices.

This in turn leads to:

- **Difficulties in effectively sharing and reusing code** in a development team.
- **A lack of effective code testing.** This can lead to errors which undermine the validity of POC results. In addition, a testing suite for the POC can provide a logical basis from which a more extensive testing suite can be constructed for the production code.
- **A failure of effectively segregating tasks** into distinct data parsing and processing, ML training and inference steps. The early segregation of code into such tasks greatly aids the later transition to the multiple pipelines which will be needed in the production deployment.

STATIC RESULTS

To prove an ML approach is working it is often necessary to apply data science techniques to static datasets as well as compare various scores and goodness-of-fit measures. However, an over-emphasis on this step risks obscuring that in production the inference will need to run on continually evolving datasets. While one is running on static datasets it is often easy to simultaneously inspect the data and the code in order to understand why bugs are occurring. On the continually changing datasets in production this is not the case. From the beginning from the POC stage, it is therefore important that the code is developed in an approach which accommodates standard monitoring and maintenance techniques of software development.

BEST PRACTICES FOR ML DEVELOPMENT

For effective development of ML applications, the approach should follow certain best practices which help avoid the PoC trap. Utilizing long-standing software engineering practices, modern approaches and state-of-the-art technologies guarantees a reliable path to production and long-term value gain.

START EARLY

It is important that production concerns are considered even at the PoC stage of the project. In many cases this can also help strengthen and validate the PoC results. Best Practices include:

- Use of **well-constructed code together with associated tests, error-handling logic, logging, and dependency management functionality**. These basic tools are crucial for allowing collaborative work on software projects and for producing reliable code.
- **Segregation of operations into logically distinct pipelines** which can be directly translated into their production equivalents if and when required.
- **Connection to the same remote data sources as used in the production use case**. If during the POC remote computational resources are utilized, these should also be aligned as closely as possible with the intended production computational environment.
- **Considering the necessary scalability of a production solution when choosing the ML algorithms** in the PoC. This avoids scalability issues and large development efforts to replace the ML strategy in later stages.
- **MVPs should be developed in such fashion that they can be expanded to a full production use case directly. Both, the code and the infrastructure should be prepared keeping the demands of future production workloads in mind.**

THE CI/CD APPROACH

CI/CD is a methodology which allows the **structured release of sequential bug fixes and new functionality features into a body of code**. The CI/CD process ensures that this is done in such way that the contributions of many distinct team members can be effectively combined, while maintaining both the local and global integrity of the code.

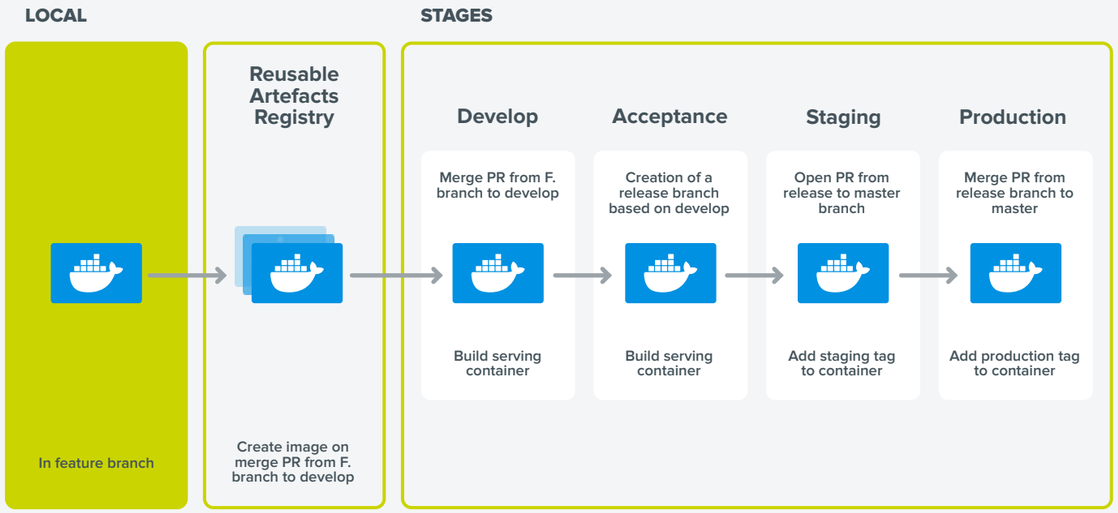
This differs greatly from the traditional script-based approach to ML development. It is however the only way to guarantee effective and constant improvements to the application and ML approach. A good CI/CD implementation always includes methods of branch management and the incorporation of various forms of automated testing, governing the deployment to the various environments.

The deployment of ML models, together with the configuration of the runtime infrastructure, should be controlled via CI/CD build and release pipelines. An example of the former operation would be the building of a docker container or image with new machine learning code and its storage in a remote antifactory. An example of the latter step would be the progressive release of this model to the different environments where training and inference can be tested.

INFRASTRUCTURE & ENVIRONMENTS

The correct choice of infrastructure early on will greatly reduce pains later in the process. Often a **container-based approach** leads to the best results in terms of automation, scalability, and fault-tolerance. This is especially true if an “infrastructure as code (IoC)” approach to resource deployment and maintenance is used. This approach manages the development of the computational resources themselves using a CI/CD methodology. This facilitates the easy establishment of **multiple environments (min. DEV, QA, PROD)**, which is a long-standing best-practice in software development and should always be applied.

ML applications should not be treated differently than other applications in this regard. As such development is necessary to **ensure that both, the algorithmic and runtime performance of the ML product is optimal**, and that it delivers the desired business value.



APPLICATION ARCHITECTURE & CHOICE OF TOOLS

The choice of an appropriate (Cloud) architecture for the ML deployment is crucial for the utility, the cost and the scalability of the tool.

Crucial components to be considered include:

- **Appropriately partitioned data-storage:** Appropriate strategies for partitioning can greatly increase the runtime performance of ML pipelines by minimizing read/write overhead.
- **Computational resources for the ML code:** The choice of appropriate computational resources depends strongly on nature of the ML problem, the approach taken to solve this problem, and the nature of stack available. Important considerations include scaling the up/down time, computational cores and RAM availability, and easy of provisioning, orchestration, and debugging. Another crucial consideration is the scaling requirements of the machine learning deployment. Scaling in this context can range in relation to the amount of data an instantiation of the ML algorithm must process, to the number of distinct ML processes which may need to run simultaneously.

- **Schedulers for the orchestration of data processing and ML pipelines:** The choice of the scheduler depends on the available stack and the level of required complexity. Triggers of machine learning jobs can be based on a variety of requirements. Thus, the range reaches from simple time- or event-based triggers to more complex conditions dependent on the success or failure of data processing and other ML pipelines. Contrasting (simple) scheduler examples are:
  - **ML deployments aimed at processing batches of data once a day**
  - **ML architectures designed for real-time inference on streaming data**
- **Visualization and/or notification services for the end users:** For the benefits of the ML deployment to fully communicate to the end user a clear and intuitive frontend or visualization service is essential. In addition, an appropriate notification service to highlight key ML results is often important.

MONITORING AND MAINTENANCE.

This topic encompasses concerns from both traditional platform and **application operations (Ops)**, and from what is now known as machine learning operations (MLOps).

Effective operations necessitate the coordination of monitoring, alarms and alerts based on the pipeline and infrastructure behaviour with effective logging and error handling within the code. The first of these features aims to quickly alert the user to pipeline failures, while the second aids the maintenance team in identifying the source of issue. These measures become particularly important as the number of running pipelines expands.

Designing the code from the ground up to accommodate such basic functionality greatly reduces the complexity of transitioning from POC to production. Conversely, **failure to adequately account for this topic can lead to extreme situations** such as waiting for hours for a job to finish, which in the end is unsuccessful and even fails to make the error visible.

MLOps relate to concerns unique to ML deployments. These include the necessity to monitor the performance of ML models and to determine when retraining is required. Such monitoring and decision making requires the effective collection, aggregation, and storage of metrics characterizing the ML algorithm performance. The choice of frameworks and infrastructure capable of meeting these requirements is an important production consideration. It may even be required to add a step to the workflow calculation KPIs for model performance.

THE ADAPTABILITY TO DIFFERENT USE CASES

Data Reply presents some concrete examples which illustrate the application of current state-of-the art ML life cycle techniques to two fundamentally distinct use cases.

MACHINE LEARNING APPROACHES IN USE

The first use case, at a global car manufacturer, is at an advanced stage of development while the second, at a large global retailer, has transitioned from MVP to production and is now being prepared for rollout in several countries.

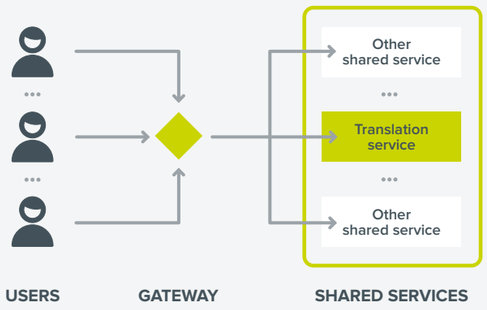
While the business cases differ greatly, the technologies and approaches used in the solution are very similar. This illustrates the adaptability of upcoming machine learning approaches based around training and inference on Kubernetes clusters.

USE CASE 1

NEURAL MACHINE TRANSLATION API WITH DOMAIN ADAPTION FOR BMW GROUP

BUSINESS CASE

In a global enterprise, translating texts is an often necessary, but time consuming and tedious task. Cutting down translation times from days to minutes with little to no manual work aids the business in working faster and more efficiently. This was achieved while providing a low-cost shared service for the entire enterprise using state-of-the-art neural machine translation models with specific adaption of the automotive domain.



TECHNICAL CASE

High quality translations required the machine translation models to be adapted to specific domain. Over the past decade, BMW's translation department collected a significant amount of technical translations and developed a prescriptive multilingual dictionary that ensures consistent usage of terminology throughout the translation. During a series of internal PoC's, the machine translation team of BMW created a methodology of injecting this data into the training process as well as enhancing the translation quality by using terminology during inference. This work resulted in trained neural models which are used in the translation pipeline.

It was necessary to deploy these models as a service for use by employees and other automated services within the company. The solution was to deploy the neural network inference code as an API. This was an example of a user-facing, real-time, machine learning service. This pattern of machine learning deployment presents several distinct challenges.

These include the need to:

- **Deploy multiple language** models responsible for the translations between different languages.
- **Efficiently redeploy after retraining** without service interruptions and with the capability to rollback if required.
- **Rapidly scale the translation service offered by a model** up and down in order to account for varying and unpredictable usage loads.

To meet these challenges, the team implemented a framework whereby models are deployed to a Kubernetes cluster using a CI/CD infrastructure based around the Seldon Core model management service. The use of this system had several distinct advantages. These include:

- **The efficient interface of Seldon Core with Helm** and its provisioning of a high-level abstraction layer to facilitate Kubernetes deployment. Its use of graph-based deployments together with the container-based nature of Kubernetes facilitates the deployment of multiple classes of models.
- This graph-based deployment also allowed for **the separation of the machine learning code and model lifecycles**. Separate CI/CD pipelines could then be built for each. This allowed for model redeployment without alterations to the production code and vice versa. Such functionality is key factor in enabling effective MLOps operations.
- **To allow rapid scaling of models to accommodate increased demand** a mixture of EC2 and Fargate containers were used on AWS. The EC2 nodes were tuned to deal with base levels of demand while the Fargate containers served to buffer for burst traffic.

“Canary Testing” was also developed in order to improve the reliability of deployments. In this feature a small percentage of traffic was routed to a newly deployed model until its functionality could be validated. When this was complete the full traffic load was redirected to the new model. If a rollback was needed this was also facilitated via the graph-based deployment approach.

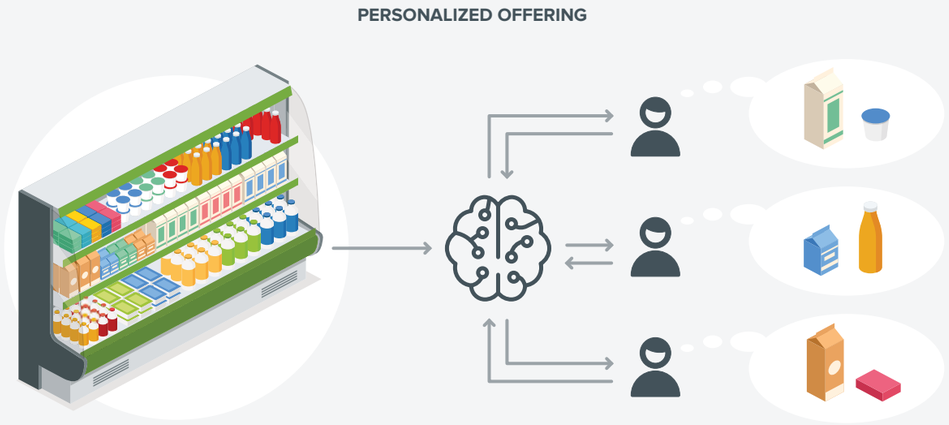
The neural machine translation API has been successfully integrated with many different automated services and workflows. First applications of the neural machine translation API have already been productionalized in 2020 revealing significant cost saving potential for the business departments.

USE CASE 2

PERSONALIZED OFFERS FOR A GLOBAL RETAILER

BUSINESS CASE

Personalized offers based on the customers’ needs, can open a huge potential for increased sales, especially in the B2B Retail business. The difficulty here lies in identifying the customer’s real needs and matching these to the product portfolio in a low-cost manner. This can be achieved through machine intelligence and automation.



TECHNICAL CASE

Being able to predict which discounted offers would lead to a successful sale with a customer meant identifying the products the customer was using and not being supplied by the retailer. This was done by applying NLP Algorithms to all available data.

The algorithms were first developed and tested on POC data before being moved to production. This production deployment consists of a multi-state pipeline for data processing, and NLP training and prediction. The key challenges were the requirements to:

- **Run scalable daily batch predictions** balancing the predictive capability of the NLP algorithms against their runtime performance.
- **Construct scheduling, orchestration, and scalable computational infrastructure** which can ingest, process, and perform predictions on the data before rapidly scaling down to minimize costs.
- **Allow for frequent, scalable retraining of models** to account for changing data. These training iterations must also be performed in a manner which aligns with the inference.

To meet these challenges, the team constructed:

- **A framework of CI/CD pipelines based around Kubeflow, Seldon-Core, and a cohesive architecture** based on Kubernetes.
- **Spark and Kubernetes in the data processing pipelines** to allow for inherently scalable pre-processing.
- NLP code using **algorithms and libraries chosen with scalability in mind.**

The product prediction service has already proven successful on data from the first European market. The capability of both the models and infrastructure to scale and parallelize effectively is currently being enhanced prior to expansion to other countries.

CONCLUSION

In this short article Data Reply has illustrated how:

- A more structured approach to machine learning development can lead to easier transitions between PoC and production.
- How the incorporation of techniques from software and big data development are becoming increasingly necessary in the machine learning lifecycle.
- How modern tools can lead to production systems which deliver consistent, long-term value.