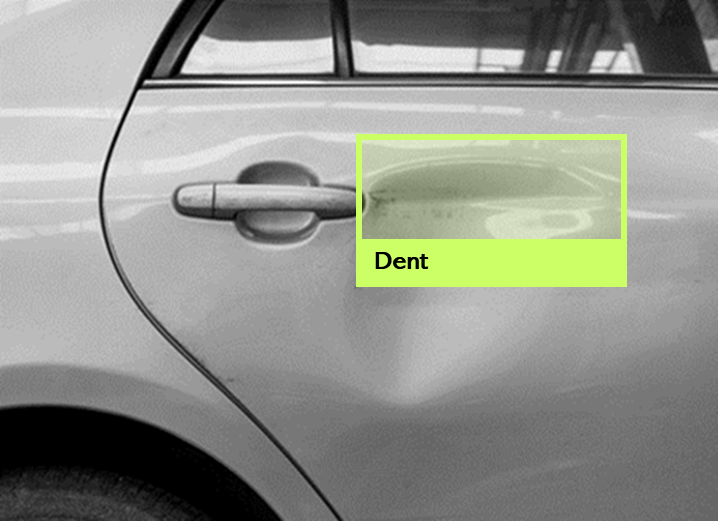
### Business goal:

Car retail company wants to provide an effective pricing strategy for any listing of cars. Earlier, this problem was solved by a manual inspection conducted by a representative, who would inspect every feature of a car. This solution was effective, as the representative considered every factor affecting the resale value (especially the damage); however, this solution could be more scalable, as the company is growing.

Using the millions of images of cars the company has collected over the years, building an object-detection model that can detect the presence of any damage in cars.

Among the various types of damage, the model should be able to detect two: scratches and dents. It should return “None” if no damage is detected.



**Model accuracy and precision**: These are traditional performance metrics used to evaluate the performance of an object detection model. Accuracy measures the percentage of correctly classified images, while precision measures the percentage of correctly classified positive (damaged) images out of all images predicted to be positive.

**Model latency**: The time it takes for the model to predict once it receives an image input. The lower the latency, the faster the model can process large numbers of images, which is important for scaling the system.

**Model throughput**: The number of images the model can process per unit of time. A higher throughput allows the system to process more images, which is important for meeting customer demand.

**Model scalability**: The ability of the system to handle a large number of requests concurrently, without significantly impacting its performance. This is important for meeting customer demand spikes and ensuring the system can handle increased traffic.

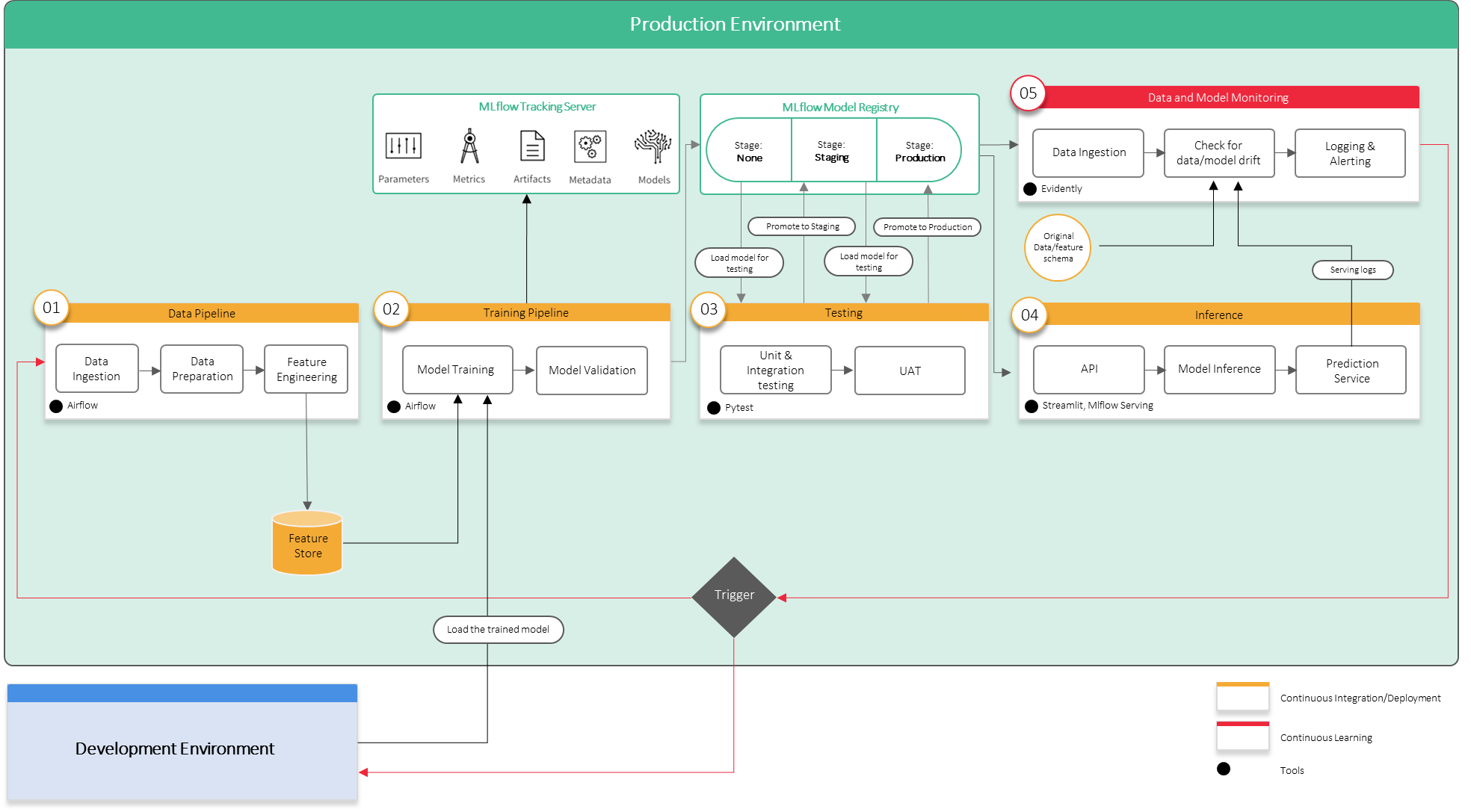
**Reduced lag in model development and deployment:** This can be achieved by acquiring and cleaning large amounts of data, tracking and versioning for experiments and model training runs and setting up the deployment and monitoring pipelines for the models that get to production.

**Experiment tracking**: This is a part (or process) of MLOps focused on collecting, organizing, and tracking model training information across multiple runs with different configurations (hyper-parameters, model size, data splits, parameters and so on). We could try different object detection models to detect the damages in a car's image accurately. Therefore We could experiment with different models, preprocessing scripts and hyper-parameters.

**Continuous training:** ML models in production can have reduced performance due to sub optimal coding and constantly evolving data profiles. As we encounter more and more data, we will encounter new diseases and treatments that may not have been present in the training data; therefore, we will need to retrain our models. Continuous training is necessary to avoid model decay over time.

**Continuous Monitoring:** The continuous monitoring pipeline will check the drift level of the input data. In case the level of drift is significant, the model will not perform well; therefore, necessary action will need to be taken to contain it.

**Real-time analytics**: Companies receive a large volume of data throughout the day. So, they have to create API’s for running models whenever clinical data arrives.



**Data pipeline:** This component is responsible for collecting and processing the raw data and preparing it for the training pipeline. In this case, the raw data consists of millions of images of used cars, some of which are not yet labeled. We will use DVC for version control, Apache Airflow for orchestration, and Amazon S3 for storage to handle this.

**Training pipeline:** This component trains the object detection model on the labeled data. We will use PyTorch as the deep learning framework and Optuna as the hyperparameter tuning library for this. We will also use Docker containers with GPU support for training, and DVC for model versioning. Finally, we will use MLflow for experiment tracking and artifact management.

**Testing pipeline:** This component tests the trained model on a holdout dataset to ensure it generalises well to new data. We will use TensorFlow, DVC, and MLflow to record the results.

**Model inference pipeline:** This component is responsible for deploying the trained model to a production environment, where it can make predictions on new data. We will use Amazon SageMaker for deployment and Amazon Lambda for serverless inference.

**Model monitoring pipeline:** This component is responsible for monitoring the deployed model to ensure it continues performing well over time. We will use Amazon CloudWatch for logging and monitoring and Amazon SNS for alerting in case of any issues. We will also use MLflow to track model performance over time, and retrain the model if necessary.

Overall, this ML system design is built to be scalable, maintainable, and reliable, with built-in mechanisms for continuous integration and continuous deployment (CI/CD) and monitoring and improvement (CMI).

**Data Pipeline:**

* Apache Spark: Apache Spark is a powerful open-source distributed computing system that is designed to handle large-scale data processing. It is a good choice for processing large amounts of data efficiently and quickly.
* DVC: Data Version Control (DVC) is an open-source version control system for data science projects. It allows us to track changes to our data and models over time and collaborate effectively.

**Training Pipeline:**

* Machine Learning Framework: PyTorch is a popular open-source machine learning framework that provides a flexible platform for building and training neural networks. It has a strong community and is well-suited for deep learning projects.
* Model Versioning: DVC is used again for model versioning, as it can track changes to models over time, enabling easy collaboration between team members and reproducibility of results.
* Model Tuning: Optuna is a popular open-source hyperparameter optimization framework. It is used to find the best hyperparameters for our models, saving time and improving model performance.
* Model Training: Docker containers with GPU support can be used to train our models on the cloud, making the training process faster and more scalable.
* Model packaging: Docker is used again for packaging our models, as it provides a portable and efficient way to deploy machine learning models.

**Inference Pipeline:**

* FastAPI: FastAPI is a modern, fast (high-performance) web framework for building APIs with Python. It is used to deploy our model as an API, making it easily accessible and scalable.
* Docker: Docker is used again to deploy our model as a containerized application, making it easy to run in any environment.

**Monitoring Pipeline:**

* Prometheus: Prometheus is an open-source monitoring system that allows us to monitor our application's performance and collect metrics about it over time.
* Grafana: Grafana is a popular open-source platform for monitoring and visualizing metrics. It is used to visualize the metrics collected by Prometheus, providing an easy way to track performance and detect issues.

**Broadly the workflow should include the following:**

* Data and model experimentation
* Automation of data pipeline
* Automation of training pipeline
* Automation of inference pipeline
* Continuous monitoring pipeline

1. **Data and Model Experimentation:**

* Collect the millions of images of cars and split them into training, validation, and testing sets.
* Explore and preprocess the data as needed to get it in a format suitable for training an object detection model.
* Use an open-source annotation tool like LabelImg or RectLabel to label the images with the presence of scratches and dents.
* Choose an appropriate object detection model, such as Faster R-CNN or YOLO, and experiment with different hyperparameters and architectures to get the best results. Tools like TensorBoard can be used to track experiments and visualize training metrics.

1. **Automation of Data Pipeline:**

* Use tools like Apache Airflow or Luigi to create a data pipeline that automates the ingestion, cleaning, preprocessing, and augmentation of new data as it becomes available.
* Store the preprocessed data in a cloud-based storage service like Amazon S3 or Google Cloud Storage.

1. **Automation of Training Pipeline:**

* Create a training pipeline using a tool like Kubeflow or MLflow that automates the training and evaluation of the object detection model on the preprocessed data.
* Use GPUs or TPUs for training to speed up the process.
* Store the trained models in a model registry, such as MLflow or Databricks Model Registry.

1. **Automation of Inference Pipeline:**

* Deploy the trained model to a cloud-based inference service like AWS SageMaker for serving predictions.
* Use a load balancer or auto-scaling group to handle varying levels of traffic.
* Set up a monitoring pipeline to detect drift in the predictions.

1. **Continuous Monitoring Pipeline:**

* Use tools like Prometheus and Grafana to monitor the model's performance in production, including metrics like accuracy, precision, recall, and F1 score.
* Use a tool like Seldon Core or MLflow Model Serving to continuously update the model based on new data and retrain it if necessary.
* Set up alerts to notify the team if the drift detected is beyond an acceptable threshold.

**Drift actions:**

1. If drift is detected due to poor lighting in the image taken, the monitoring pipeline will be triggered. The monitoring pipeline will alert the team to investigate the issue and retrain the model with additional images with poor lighting conditions.
2. If additional annotated data is available, the data pipeline will be triggered. The data pipeline will automatically ingest and preprocess the new data, which will be added to the training set. The training pipeline will then be triggered to retrain the model on the augmented dataset, and the new model will be deployed to the inference pipeline for serving predictions.