Argus

Release 1.0.0

Javier De Velasco Oriol

Dec 18, 2022

WHY ARGUS

Argus is a Proof Of Concept for Computer Vision ADAS model validation and evaluation, based on an adversarial approach.

Argus is:

- Versatile
- Explainable
- Scalable
- Statistically Robust
- Cost-Effective
- Easy to implement
- Constantly improving

This chapter explores the reasoning behind the development of Argus, the challenge it aims to solve and the academic inspiration that supports it.

The main source of inspiration is based mainly on Multi-weather city: Adverse weather stacking for autonomous driving by Musat et al

CHAPTER

ONE

ADVERSE-WEATHER IMPACT ON ADAS

Adverse weather conditions, such as heavy rain, snow, fog, or sandstorms, can significantly impact the performance of camera-based Advanced Driver Assistance Systems (ADAS). These systems rely on cameras to capture images of the surrounding environment and use computer vision algorithms to interpret and analyze the data.

In poor weather conditions, the visibility of the camera may be reduced, which can lead to a decrease in the accuracy and reliability of ADAS systems. For example, if the camera is unable to clearly see the road or other objects due to heavy rain or snow, it may not be able to accurately detect and classify objects, leading to a higher rate of false positives or false negatives. Adverse weather conditions can also cause reflections on the camera lens, which can distort the image and make it difficult for the ADAS to accurately interpret the data.

Adverse weather can significantly impact the performance of camera-based ADAS systems, leading to a decrease in their effectiveness and reliability. It is important for developers and engineers to be aware of these limitations and create systems that correctly function under such adverse conditions.

1.1 Main Adverse-Weather conditions

These conditions (not fully comprehensive) can negatively impact the performance of cameras:

- Night: Low illuminance, reduced detection range, reduced color range, high contrast from other vehicle lamplights.
- Cloudy: reduced illuminance.
- Rain: reduced illuminance, reflective areas,stochastic localization of puddles.
- Snow: high contrast, occluded markings, stochastic localization of snow.
- Sandstorm: very low visibility.
- Smoke: very low visibility.
- Fog: diffuse low illuminance, highly diffused from other vehicle lamplights.

Additionally, the camera sensor can be blocked /occluded by:

- Raindrops
- Mud
- Snow

1.2 Adverse weather combinations

Weather-generated conditions can be combined between each other (Rain droplet occlusion can be present or not on a rainy day)

Due to the nature of weather effects, some of them are exclusive with each other (One cannot have rain falling on a clear day,) which limits the combinatorics explosion and simplifies the simulated feature space.

Additionally, some phenomena are limited geographically or temporally (a client selling vehicles only in Dubai is unlikely to need evaluation of snow environments in the same way as a Norwegian company won't need to test sandstorms)

1.3 Impact on ML

As ML models are based on analyzing data from a given input source, if this input source differs from the expected data that it was trained from it has the potential of misclassifying / miscalculating the desired output, and can cause a downstream error due to an erroneous classification leading to an incorrect action being taken.

The increasing dependency (based on the usefulness of ML) of ADAS systems on ML-based systems opens this avenue for potential failure into the system and as such its highly desirable to make the ML models as robust as possible

1.4 Solutions

Hardware-based solutions are one partial, possible solution for this. By making sure the input is as clean as possible, downstream errors due to erroneous inputs can be avoided. This cannot solve all possible adverse weather phenomena as some can't be fixed within the physical boundaries of the vehicle (such as snow covering a lane marker). Alternative sensors are also explored as viable solutions for some of these problems (such as Radars and Lidars) but some present their own challenges, and tend to be more expensive.

Software-based solutions are thus being explored. The main issue with ML models is that they are intrinsically limited in their decision-making process by what they have been trained upon. Thus if a ML model has never been trained in adverse-weather scenarios its performance in the real world under those conditions cannot be known beforehand and the odds of an issue occurring are likely. Thus the main software-based solution has been to increase the training dataset (as well as related evaluation and testing) to also include these scenarios and thus preemptively select those models that perform well under all conditions.

The problem arises due to the fact that obtaining this data is nontrivial and expensive. Most of these conditions cannot be predicted beforehand, and occur on limited dates thus complicating the data-collection process at sufficiently large scales to correctly compensate it (this also exacerbates the class-imbalance problems of ML). Thus the dataset-collection part of ML becomes the bottleneck to solve.

Simulation has been constantly proposed as a plausible alternative to generate synthetic data that is real-like and can be used to artificially augment the datasets to enhance it and compensate for this bottleneck(it comes with its own set of challenges, which will be further explored).

CHAPTER

TWO

PREVIOUS RESEARCH

This will mainly analyze the findings of Multi-weather city: Adverse weather stacking for autonomous driving by Musat et al. A more thorough and comprehensive analysis of the academic literature is desired though.

2.1 Scope

The authors propose a GAN-based simulation pipeline to generate synthetic images from clear images that have multiple variations based on different adverse weather conditions (Thus obtaining 7 synthetic images from a given real image)

The weather conditions to simulate are rain, night, snow, and rain droplets (plus the mix of droplets+one of the other conditions).

2.2 Main Findings

The authors prove that a modular GAN-based synthetic data simulation pipeline can generate useful synthetic data from clear weather images and that this synthetically-augmented dataset can then be used to improve AD tasks.

The use of multiple generated conditions for training rather than a single generated condition gives the best results in OD and segmentation tasks.

2.3 Datasets

The generative networks are trained on the Cityscapes dataset, as well as Oxford RobotCar, Dark Zurich and, RainyScreens. Then, they are evaluated in the Mapillary, BDD100K, DAWN and ACDC datasets to evaluate how useful are GAN-generated synthetic datasets at improving OD and segmentation tasks.

The datasets present an interesting challenge as the number of samples in some particular combinations (hence highlighting the dimensionality/ data acquisition challenge this actually wants to solve)

2.4 Evaluation

The synthetic data generated from the GANs are evaluated in terms of quality with the Frechet Inception Distance (FID) and Inception Score (IS), and quantitatively by using pre-trained OD and segmentation models (Detectron2) fine-tuned with the resulting synthetically-augmented Cityscapes dataset and then evaluated with the out-of-distribution testing datasets

2.5 Results

The results from the qualitative testing show that the synthetically generated images that results from applying the trained GANs on the testing dataset show potential at detection and segmentation tasks.

The results from the quantitative show that the fine-tuned synthetic datasets for a particular weather condition do not see an important increase w.r.t. the baseline, but that when fine-tuning with all conditions it improves well enough. This suggests that the single-condition overfits towards that condition and worsens the other ones. This also seems to have a beneficial effect in the baseline too, obtaining better results when testing on the cityscapes test and using the synthetically-augmented dataset for training.

Additionally, the results from both tests are consistent with each other as low (good) ranking FID classes also perform well in the actual OD and segmentation tasks, thus reinforcing the underlying hypothesis.

2.6 Notes

The mapping of the droplet adherent is limited to being trained on $(clear, overcast, daytime) - > (droplet, overcast, daytime)$ and is then used for $(clear, X, X)$. A more thorough solution would be to obtain each mapping for each weather/lightning condition, although this risks an exploding complexity. Alternatively, the GAN can be trained on a diverse dataset that consists of non-droplets and droplets from multiple weather/lightning conditions (ideally balanced). This can be difficult to obtain due to the aforementioned dataset bottleneck

Care should be taken to implement the pipeline completely to each image in the desired dataset, as differing copies per image could cause the augmented dataset to have overrepresented samples that are overfitted towards.

Using the pre-trained Detectron2 (already trained on Cityscapes) and then fine-tuning it to Cityscapes risks overrepresenting samples (but would not impact the testing results as it is done with out-of-distribution samples, i.e. no a priori bias)

CHAPTER

THREE

DIMENSIONALITY EXPLOSION

In the context of machine learning, dimensionality explosion, or commonly the curse of dimensionality, refers to the phenomenon of increasing the number of dimensions (or features) in a dataset to such a large extent that it becomes difficult or impossible to effectively analyze or process the data. This can occur when the number of dimensions in a dataset is much larger than the number of samples, which can lead to problems such as overfitting, poor generalization, and increased computational complexity.

To address dimensionality explosion, it may be necessary to use techniques such as feature selection, dimensionality reduction, or regularization in order to reduce the number of dimensions in the dataset and improve the performance and scalability of ML models.

3.1 Implications

Within the scope of Argus, the dimensionality explosion arises from two areas:

- The increasing combination of possible adverse weather scenarios, and their relatively sparse frequency in the real world that significantly reduces the availability of real-world data(which typically leads to class imbalance and complexities in the model training). This makes it so that the higher-dimensional space of weather-combinations is sparse and thus makes training of generator models that can map this domain shift more complex (data needs to be taken of these low-frequency situations).
- The incredibly expansive operational space of an ADAS function(practically infinite as driving in the real world is a N-dimensional feature space in practically infinite scenarios). An advanced (L4/L5) ADAS ML model in production can potentially be expected to be in use under most of these scenarios (granted, the scenarios are clustered within this very sparse space and as such can be easily evaluated). As a validator of ADAS models Argus needs to be able to evaluate these scenarios up to a certain degree to satisfy some statistical certainty of a sufficiently well-performing model under diverse conditions. This is a general challenge of the field that is present in any ADAS solution for practical matters.

3.2 Solutions

There is not a silver bullet to solve these issues (particularly the second, if there was, autonomous vehicles would already be operating in all cities, rather than in a few selected ones). This then needs to be tackled by alternate methods that are smaller in scope.

For the first scenario it is then perhaps possible that the frequency within the dataset matches the expected frequency in the real world (i.e. matching the probability distributions), and a weighted metric of evaluation is then used for evaluation. The generator can then try and compensate with a perhaps limited dataset that works well with the weighted metric, and the . Additionally, statistical certainty can thus guide to an adequate number of scenarios for the generator dataset.

An interesting avenue of how this is solved is in the way nature and us humans by proxy solve it, by quickly learning from single experiences from a shared universal model and efficiently extrapolating it to unseen scenarios (i.e. I slip on a frozen floor and then know that I must be careful the first time I drive on snow).

CHAPTER

FOUR

SIM-TO-REAL GAP

The Sim-to-Real gap refers to the difference between the performance of machine learning models in a simulated environment versus their performance in the real world. This gap can be a significant challenge when developing and deploying machine learning models for a variety of applications, including autonomous systems such as self-driving cars.

One of the main causes of the The Sim-to-Real gap is the fact that simulated environments often differ significantly from the real world in terms of their appearance, dynamics, and sensory data. This can lead to machine learning models that perform well in simulation but poorly in the real world, due to the inability to accurately generalize from the simulated environment to the real one.

Commonly, the The Sim-to-Real gap is addressed by a mix of the following:

- Realistic simulations: By using simulations that more accurately capture the appearance, dynamics, and sensory data of the real world, the models thus learn closer to reality and can thus perform better. The probabilistic distribution of the simulation matches more closely the one seen in the real world.
- Domain randomization: Domain randomization involves adding random variability to the simulated environment in order to better approximate the complexity and variability of the real world. This can help machine learning models to learn more robustly and generalize better to the real world.
- Transfer learning: Transfer learning involves training a machine learning model on a large, diverse dataset and then fine-tuning it on a smaller, more specific dataset. This can help the model to handle a wide range of variations and scenarios.

Addressing the The Sim-to-Real gap is a critical challenge when developing and deploying machine learning models for applications such as autonomous systems, and requires a combination of careful simulation design and robust machine learning techniques.

4.1 How we tackle it

Argus aims to reduce the The Sim-to-Real gap by maximizing the similarity from generated adverse weather conditions to real world conditions. This is accomplished by using highly-performant GAN models that maximize the nearness intrinsically. The best way of ensuring models are robust to be used in the real world is by ensuring they are both trained and validated in scenarios that are highly diverse and representative of the real world.

Argus ensures your model generalizes well to the different real-world conditions, and generates necessary data to train when it doesn't. It also ensures that any model that is over-performing on simulation is detected, by comparing it to real-like data. This has the benefit of reducing the likelihood of delivering to production a promising model in simulation that translates badly to real-world scenarios.

CHAPTER

FIVE

GANS AND CYCLEGANS

Generative Adversarial Networks (GANs) are a type of machine learning model that is made up of two neural networks: a generator network and a discriminator network. The generator network is trained to generate new, synthetic data that is similar to a dataset that it has been trained on. The discriminator network is trained to determine whether a given sample is real or synthetic, based on the dataset it has been trained on.

During training, the generator and discriminator networks are pitted against each other in a two-part “game.” The generator network tries to generate synthetic data that is similar enough to the real data that the discriminator network can’t tell the difference, while the discriminator network tries to correctly identify whether each sample is real or synthetic. As training progresses, both networks become better at their respective tasks, until the generator network is able to generate synthetic data that is virtually indistinguishable from the real data. This is represented in the following equation as a minimax game:

$$\min_G \max_D \vec{E}_{x\sim P_{\text{real}}}[\log D(x)] + \vec{E}_{z\sim P_G}[\log(1 - D(G(z)))]$$

CycleGANs (short for Cycle-Consistent Generative Adversarial Networks) are a variant of GAN that are specifically designed to perform image-to-image translation tasks. In other words, CycleGANs are used to transform images from one domain into images in another domain while maintaining the same content and structure. An example of this would be transforming an image under clear weather conditions to one the same one as it would appear under adverse weather conditions (yet maintaining the environment data)

One of the key features of CycleGANs is their use of cycle consistency loss, which helps to ensure that the generated images are not only visually similar to the target domain, but also preserve the content and structure of the original images. This is achieved by training the CycleGAN to translate images back and forth between the two domains and using a loss function that measures the difference between the original image and the translated image.

5.1 Usage

Within Argus, GANs are used in the generation phase of the model evaluation pipeline. Multiple GAN models are trained by the Argus team, and are then used in the dataset augmentation process to generate the adverse weather data for model evaluation. These models are constantly updated based on new data and then evaluated to always provide the best possible results that correctly match. You can select multiple models according to your needs, thereby augmenting your dataset to be augmented with hard-to-obtain adverse weather images and enabling a more robust model validation at a scale.

One of the key benefits of GANs and CycleGANs is that they help tackle the Sim-to-Real gap innately thanks to is discriminator/generator architecture. A GAN with a robust dataset that adequately converges will by definition maximize the closeness of simulated images to real-world images, thus reducing the gap and ensuring the models are validated with real-like data(or the synthetic images can be used for training and thus the model is trained to match real-like probability distributions).

5.2 Cost-effective

GAN's that have adequately learned the domain mapping from a "clean" image to a adverse weather one (whichever effect it is), have the benefit of being more efficient and cost-effective than phenomenological simulation, given that the underlying probabilistic model that has been trained is sufficiently representative of the actual changes due to the physical phenomena.

Effectively, the GAN model can bypass the compute-intensive calculation by "learning" an efficient (i.e. densely encoded) mapping of the domain shift in its training step, which can then be easily ran in a new image or scenario at a low computational cost. The phenomenological model necessarily needs to run compute-intensive tasks for each single frame every time it is run. When the number of frames / recordings needed lies in the millions, it might make more economical sense to use the pre-trained GAN.

CHAPTER

SIX

EVALUATION

To properly prove that the generative models that are used in the Argus system are indeed real-like and a useful tool for evaluation of ML models they need to be thoroughly evaluated and compared to other models. The synthetic images should be evaluated both qualitatively and quantitatively to ensure that they are indeed real-like and any models that use those images for evaluation can be expected to perform similarly in the real world. This is especially important for safety-critical systems in ADAS and autonomous vehicles. If this cannot be proved with statistical certainty, then the results of the evaluations will not transfer well to the real world(Sim-to-Real gap) and the use of this system for validation is then worthless. The evaluation follows the procedure outlined in Multi-weather city: Adverse weather stacking for autonomous driving by Musat et al as it seems robust enough.

6.1 Qualitative Evalution of Generative Models

To evaluate the synthetical augmentation of datasets the generated output from the adverse weather GAN stack can be evaluated with the Frechet Inception Distance and Inception Score. These two metrics give an estimate of the image quality and its suitability for evaluation in OD and segmentation tasks. A low FID score and a high IS would represent a good synthetic image. Images that don't fulfill this criteria can be marked for further evaluation as images of interest that can guide future improvements in the generative model.

6.2 Quantitative Evaluation of Generative Models

To quantitatively evaluate the synthetically augmented-dataset this dataset is then used to train multiple reference Evaluation ML models that perform specific ADAS tasks such as OD and segmentation. Then, the model is tested in Out-of-Distribution samples(for example a different dataset) and its performance is contrasted versus the real-only trained model. By evaluating the performance of the downstream tasks the validity of the synthetic augmentation can be confirmed if it helps improve the evaluation metrics on the test dataset w.r.t the real-trained. If both the quantitative and qualitative metrics show correlation this doubly confirms the validity of the generative model at correctly learning the domain shift and thus implying the synthetic images are real-like (and thus representative of a real-world test). A model that performs better over multiple evaluation models(provided a diverse and balanced testing dataset with the desired adverse weather conditions) has a higher statistical certainty and can be expected to perform well on the real world.

6.3 Evaluation of the actual models

The main goal of Argus is to provide ML model evaluation using synthetically augmented datasets to test the performance in multiple adverse weather conditions (which is hard to accomplish in traditional datasets). For this reason the ML models need to be thoroughly evaluated with multiple metrics that accurately describe the performance of the model for the given task using the synthetically augmented test dataset. Depending on the scope of each model is the evaluation scheme that will be used (i.e. IoU for segmentation, AP for object detection, etc.). All of the ML model testing must be done using out-of-distribution images,both the synthetic as well as the source/real (or risk erroneously overperforming metrics due to a priori biases). Additionally, the synthetically generated dataset should be balanced in its creation to ensure the different adverse weather conditions are applied equally and there is no overfitting due to a mismatched oversampling of specific scenarios.

CHAPTER

SEVEN

ARCHITECTURE

This chapter briefly explores the architectural implementation of Argus, its tradeoffs, challenges and its design implications.

7.1 Datasets

Datasets are the core of any ML system. They are the source of truth for the system to learn from, and thus the quality of the dataset is the most important factor in the quality of the system. Datasets are also the most expensive part of the ML system, as they are the most time consuming to collect and the most costly to obtain. It is important to ensure that the dataset is representative of the problem that the model is being trained to solve, and that it has been cleaned and preprocessed to remove any errors or inconsistencies. Thus there is a pressing need to use a clean, representative and large ADAS dataset for training and validation of generative and evaluation models. The selection of datasets is crucial for the success of Argus.

7.1.1 Open Source Datasets

Open Source datasets are mostly useful for Proof of Concept and simple validation that the system works correctly (and thus would work with a proprietary dataset). Most datasets that are open source are made available under non-commercial use terms, and as such can't be used in products that are meant to be sold

Some of the OS datasets that can be used for research purposes are the following:

- Cityscapes
- KITTI
- BDD100K
- NuScenes
- A2D2
- Apolloscape
- comma2k19
- ImageNet

7.1.2 Proprietary Datasets

The use of proprietary datasets is ideal from a legal standpoint as there are no limitations to its financial usage, but are complex to obtain and are highly costly. A fleet of vehicles needs to be setup with sensor rigs and those vehicles then need to be driven in diverse environments. The dataset is then restricted by the locations and conditions that are accessible to the fleet of driving vehicles. In multiple countries recording vehicles need to obtain legal rights to record their environment and can be limited in the scope of the data collected.

7.1.3 Client Datasets

The use of datasets that clients have collected is promising as it offloads the cost and legal implications of dataset collection unto the client, but for the same reasons is the most complex to obtain. Its highly unlikely that a client will allow unrestricted access to their dataset for use by another party (as datasets grant a competitive advantage), and sometimes the legal implications straight do not allow this.

7.1.4 Third-party Datasets

Third party datasets can also be explored, but those definitely involve an economical cost. Still, it might make financial sense to use a third-party dataset if its cost is smaller than the cost of collecting the data yourself.

7.1.5 Simulated Datasets

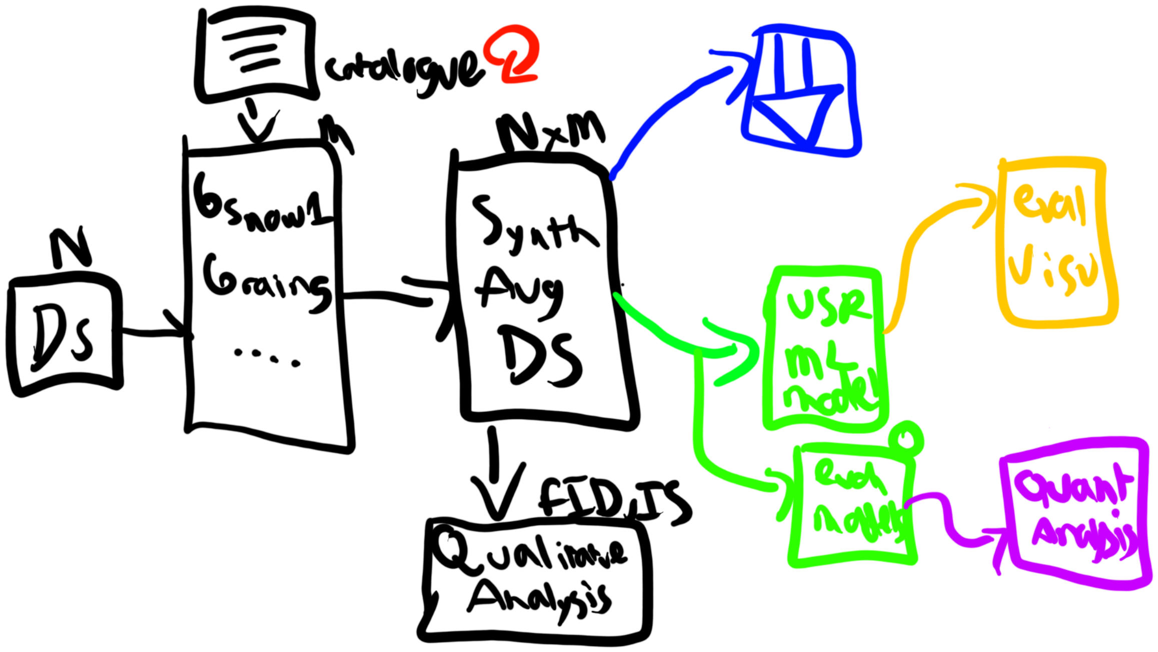
Tools like Nvidia DRIVESim and Parallel Domain can be used to generate synthetic datasets that can be used for training and validation. The main advantage of using synthetic data is that its cost is much lower, it tends to be deterministic and it can be generated in a controlled environment, and thus can be used to test the system in scenarios that are not possible in the real world. The main disadvantage is that the data is not real, and thus the system might not generalize well to the real world depending on the simulation engine.

There is a risk that using simulated datasets for the generative models won't really cross the sim-to-real gap, as it can generate erroneous a priori biases in the data that match the simulation and erroneously appear to be reliable. Thus, simulated datasets should be used as a last resort.

7.2 Stackable Plug & Play Generative Model Architecture

The principal idea from Musat et al of using a modular weather-stacking for synthetic data generation is quite useful and shows promise. The results from their publication appear to show that the total mix of all adverse weather combination is the best performing, nevertheless there can be potential scenarios when a limited subset is of use, or when only specific conditions are necessary in specific regions. Thus a flexible architecture that can be mix-and-matched can have potential and is worth exploring. Argus aims to use this idea as a central building block of a validation architecture for ML models.

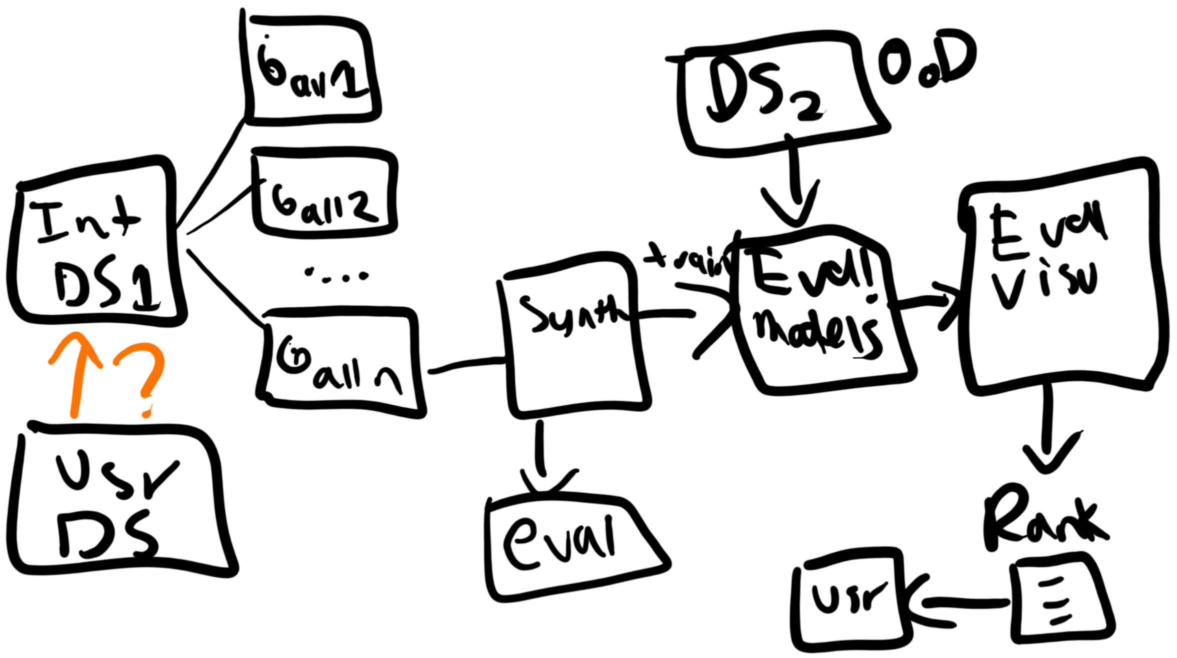
A rough diagram of the desired flow architecture is as follows:



The flow architecture is composed of the following components:

- User provided datasets (either for augmentation or for validation)
- A stack of possible GAN-based generative models that can be mixed together (per Musat et al) to generate synthetic data. These models are already evaluated and validated in a catalogue and the client can select the ones that are useful for their use case based on the results (transparency). They are constantly updated and evaluated by the Argus team.
- The dataset is synthetically augmented with the chosen generative stack (or the default one). Qualitative evaluation is performed so the client can trust the generated images will be real-like
- The user can download the synth data for train dataset augmentation, or their custom ML model can be evaluated with the given synthetically augmented dataset. The results are provided to the user and visualizations follow to raise critical points & issues and make it very clear where the model is performing correctly and where it isn't.
- At the same time, the synthetic images are also evaluated with the internal evaluation models to analyze the performance of the generative models and refine them if necessary.

For the internal team the desired flow architecture is shown below:



The main benefits of this approach are:

- Versatility: Enabling the use of multiple stacked weather conditions increases the potential dimensionality of the synthetic data that can be generated. This then allows the client to generate synthetic data for testing in wider range of use cases, or potentially larger training dataset.
- Validation Robustness: By using a modular approach, we can ensure that the generative model can generalize well to the different weather conditions that are expected in the real world. This then allows validation of the desired ML model in multiple adverse weather conditions and gauge if it can how well it can possibly generalize in the real world.
- Cost-efficiency: By learning an efficient mapping of the domain shift the generative approach can generate simulated data from multiple weather conditions in an efficient manner which translates to a lower cost (when compared with phyiscs-based simulation).
- Edge case generation: Many adverse weather conditions are complex to observer in the real world, and mixes of them can be even less frequent, as such a system that can potentially stack them and extend previously seen normal weather conditions can be useful for edge case generation that can potentially cause critical errors.
- Explainability: The generative approach can be used to generate synthetic data that can be used to explain the model's behavior in different weather conditions. This can be useful for debugging and understanding the model's behavior in different conditions.

On the contrary, the challenges that this approach presents are as follows:

- Dimensional complexity: When the number of adverse weather conditions, the potential combinations increases proportionally. As the dimensional space increases in size, data will thus be more sparse and harder to obtain. This is a challenge when obtaining the dataset to train the generative models on.
- Dataset: To properly evaluate GAN models, balanced and representative datasets must be obtained that closely match the probability distributions of the real world. This can be costly and time consuming. As better generative models are desired, the requirements on the dataset increase to ensure the models can keep on improving.
- Training reuse: While generative models that are worthwhile can be used multiple times to generate synthetic data for multiple datasets, the training process is still costly and time consuming. Continuous development of

multiple generative models can thus be costly particularly if specific combinations are not used frequently. Thus care needs to be taken to evaluate the tradeoffs between flexibility and usefulness.

7.3 Infrastructure Devops, & MLOps

7.3.1 Challenges and risks

The main challenges and risks that the Argus system has:

- Safety: The safety of ADAS systems is paramount, and as such every decision taken with the system must begin by being thoroughly analyzed and its implications must be assumed to be robust and accurate enough as there is potential of risking human lives in the case of a failure.
- Proper Dataset Selection: The generative models are only as good as the source dataset that is being used to train them. For this reason the dataset needs to be constantly improved, analyzed and validated. Diverse adverse weather conditions need to be obtained, and this can be complex due to the legal, technical and climate conditions. The dataset acquisition can be tackled as described in the datasets and data-flywheel sections.
- Model Creep / Combinatorics / Complexity: by mix-and-matching multiple adverse condition generative models the system runs the risk of increasing the complexity of the system and the number of models that need to be trained and maintained. This is even more relevant when it is considered that the outputs of the models then need to be evaluated with multiple downstream task models. A balance must be struck to ensure optimal performance at a functional scale.
- Scalable architecture: The architecture must be able to grow and scale successfully to truly accomplish the task Argus aims to fulfill. A dependable testing/ validation system needs to be highly reliable to successfully empower the reality of Autonomous Driving in a safe manner.
- Model architecture Lock-in: Care must be taken to explore other architectures for generative models, as its possible that the proposed solution is not the ideal one, and a desire to see it work out must not prevent from analyzing it logically.
- Diminishing returns: Its possible that at some point the training of generative models will not be able to improve the performance of the system, and the effort to train them will not be worth the gains. This is a risk that needs to be mitigated by constantly evaluating the performance of the generative models and the downstream task, and compare them with the running costs and improvements seen.
- Economics: The development must always take into consideration the economics of the system, as its a for-profit venture and as such needs to be financially viable for its continuous development. Far-fetched solutions risk being too expensive to implement and maintain, and as such must be avoided. Efficiency must be a priority.

7.3.2 Infrastructure & Devops

The Argus system needs to be deployed and available for multiple clients as a SaaS-based solution. This implies availability and highly responsive user experience for the different clients that can scale.

A scalable infrastructure architecture can be implemented by using cluster-based Kubernetes. Dockerized environments can be used for internal development and releases at the prototyping phases. Prometheus can be useful to monitor containers and clusters to identify bottlenecks and errors. A Load Balancer such as ELB, ALB, CLB or Nginx can be used to balance the load between multiple clusters efficiently if the client load is high enough. This system can be deployed on a cloud-based infrastructure, such as AWS, GCP, or Azure, to ensure scalability and availability or alternatively on a self-hosted infrastructure, such as multiple on-premises servers, to ensure security and privacy. Each of these has particular tradeoffs that are explored further on.

In terms of development of the codebase, Github/ Gitlab have been consistency used for version control and collaboration, and most developers are familiar with these tools minimizing the need for trainings. Jenkins can be used for

Continuous Integration and Continuous Development. Unit testing and test automation can also be implemented to ensure errors are quickly identified.

7.3.3 MLOps

Additionally, as Argus is a ML-focused solution, MLOps is a highly crucial area that can potentially make or break the system. A proper MLOps pipeline can greatly reduced the effort in, enabling the developers to focus on creative generative model design, and insight validation/ evaluation rather than the nitty-gritty ops work. Handling the training, validation, testing, logging, release and analysis in the background is highly desirable. The main areas where MLOps can empower the Argus system are:

- Distributed Computing
- Experiment Tracking / Logging
- Model Registry
- Model Deployment
- Metadata Storage
- Visualization

Apache Spark, Hadoop or dCUDA can be used to distribute computation-intensive tasks across multiple instances. Tensorflow, Pytorch and Elephas (by leveraging Spark) can also be used to distribute the ML training phases into multiple devices if required. This is particularly crucial if the generative models need to be trained on a large ADAS dataset to ensure the performance of such models is highly powerful. Additionally, retraining the evaluation models can also be particularly expensive (multiple models)

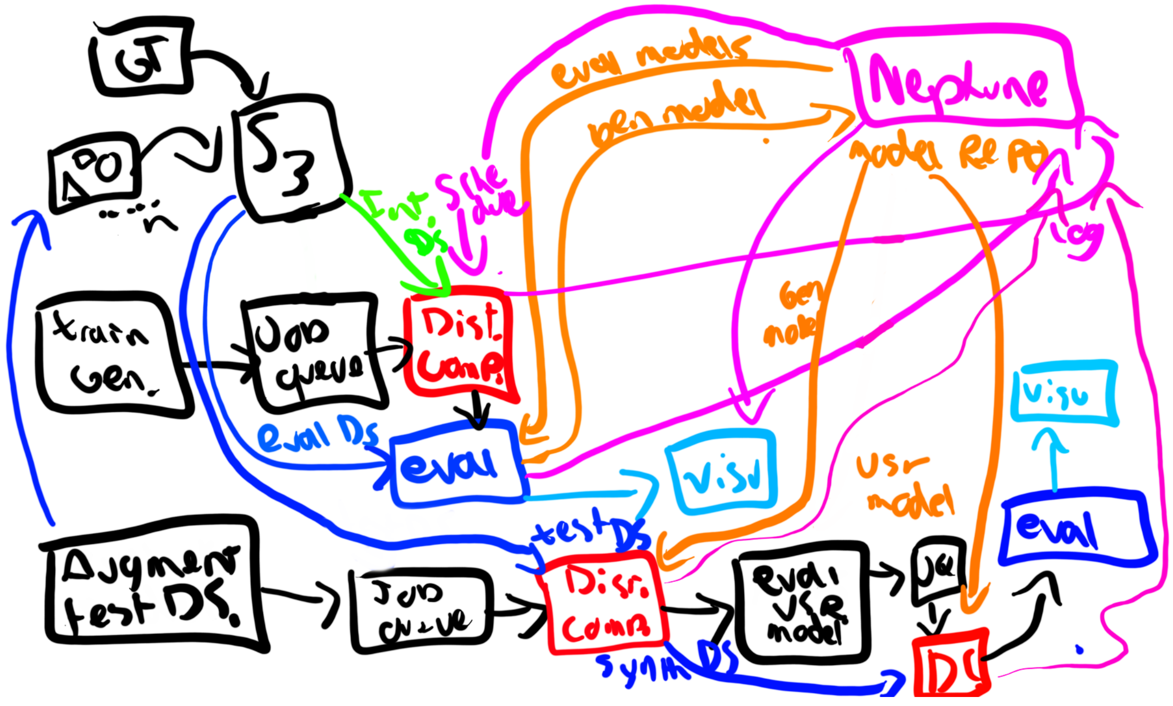
Training the generative and evaluation models is a complex task that is best developed in tandem with an Experiment Tracking tool that allows orchestration of ML model experimentation and logs it for further processing. Some possible tools for achieving this are Neptune, Tensorboard, Weights and Biases, ClearML, MLFlow, Comet, and Polyaxon.

Metadata tracking is quite useful as it allows the system to group frames based on specific data which might be useful for further validation. For example a recurrent set of difficult to simulate frames can be grouped together and used to focus the development performance of subsequent generative models. This can be done by using a metadata storage system such as MLFlow, Neptune, or Vertex.

Visualization of the training and testing results is highly useful for continuous development and improvement of the model by visually showcasing the historic and current performance of models at a simple glance, and by allowing the user to quickly identify frames that might be problematic. This can be done by using a visualization tool such as Tensorboard, MLFlow, or Neptune.

By leveraging multiple technologies, development of the Argus system can be greatly empowered and the development time invested in routine areas can be reduced significantly. This is particularly useful as it allows developers to fully focus on the generative models as they are the most complex and time-consuming part of the system, allowing a constant stream of updates and better generative models.

A sample architectural diagram of the infrastructure solution is shown below:



7.3.4 Cloud vs Self-Hosted Infrastructure Tradeoffs

Implementation of Self-Hosted infrastructure is a viable option, but it requires a lot of time and effort to maintain and scale. It is also more expensive than using a cloud provider due to volume efficiencies. The advantage is that it gives total control over the hardware and software, and as such can be more flexible if needed. Additionally, it is more secure than cloud providers, as it is not exposed to the internet and can be isolated from the rest of the network. Furthermore, financially, it can be cost-effective in the long run, as the hardware investment cost can be recovered over time.

The use of S3 or similar storage systems allows offloading the data handling to a scalable on-demand solution that can potentially scale as much as it needs to. Datalake structures can be used to allow multiple sources of information to be used for different models or components within the company. If the training is done with on-premise devices, then downloading the datasets can have a high data transfer and as such would be much more expensive, in this scenario with local training a high-speed SSD / NVME is ideal to ensure the R/W is not the bottleneck of the system. Dataset regulations might make Self-Hosted necessary, if the law identifies that particular datasets that are sensitive cannot leave the country.

Cloud-based providers in contrast have very strict SLAs and uptime requirements. They are also easier to maintain and do not require a dedicated crew to maintain the infrastructure. They are also more flexible, as they can be scaled up and down as needed based on demand. Additionally, support is readily available and the risk of downtime is mitigated.

GPU-based training on cloud can be be expensive, but requires a much smaller upfront investment as the cost is amortized over a larger number of users, and can be scaled much more easily than setting up a new GPU cluster. The cost of training on cloud is also lower than on-premise particularly when the electricity costs are high, and the maintenance costs also tends to be much lower.

7.3.5 MLOps-as-a-Service vs Self-Hosted MLOps

The main downside of MLOps-as-a-service tends to be the cost associated to it, particularly when it scales based on the number of requests / runs that are necessary. As the number of models grows and their combinations the cost will then tend to increase. The internal development of MLOps can also be achieved with effort from the internal team, although this has the downside that time and effort must be dedicated to it, when it can be better spent working on the generative models which are the main competitive advantage. A MLOps system that is both nimble and flexible, and can be adapted to the needs of the development team is ideal for bootstrapping the Argus system in its early stages. Upon further success it can be reevaluated if migrating to a proprietary pipeline is better. A complex pipeline also adds complexity to the system so a single provider could be used at a early stage to minimize the risk of complexity.

7.4 Data Flywheel

In the field of machine learning, a data flywheel refers to a self-sustaining cycle of data collection, analysis, and use that drives continuous improvement and growth in ML models and systems. This might involve collecting data from a variety of sources (such as sensor data, customer interactions, or product usage data), using that data to train and refine ML models, and then using those models to make predictions or decisions that drive further data collection and analysis.

One key aspect of a data flywheel in the context of ML is the way in which it can drive the continuous improvement and evolution of ML models and systems. By collecting and analyzing more data, ML models can become more accurate and reliable over time, leading to better predictions and decisions, which in turn drives the collection of more data. This creates a virtuous cycle that can drive sustained improvement and growth in the performance and capabilities of ML models and systems.

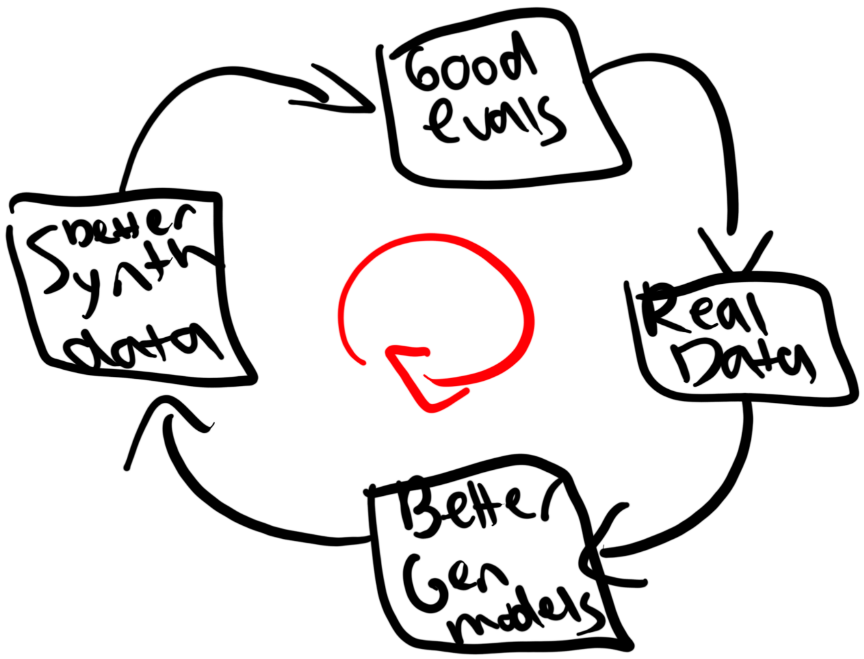
Data flywheels can be used in a variety of applications, including predictive analytics, recommendation systems, and autonomous systems. In each case, the goal is to use data to drive continuous improvement and growth in the performance and capabilities of ML models and systems, creating a self-sustaining cycle that drives long-term success.

7.4.1 Usage

Withing Argus the data-flywheel approach can be implemented to ensure that each use of the tool further expands the potential of itself to provide the best results.

The usefulness of Argus as a worthwhile product depends primarily on its potential to provide the best possible evaluations for the client models (and a better user experience). As the validation is mostly propelled by the GANs and CycleGANs, having the best-performant GANs is crucial. This can be achieved either by a better, more-diverse dataset, or by unique architectures.

The flywheel tackles the first part by ensuring the GAN models are continuously trained on an ever-expanding, more diverse dataset to ensure the best models are selected at every step of the way, and thus more clients are using it(providing more useful information that then improves the models and so on ad infinitum).



7.4.2 Implementation

To implement the "flywheel", periodic training and evaluation of the multiple GAN architectures on increased datasets should be performed, thus selecting the best trained models for client use. Client usage (when applicable) of previously unseen images should be fed into the generator dataset (with proper class distribution care). In the same vein, obtaining more real world data also allows extending the training, validation and testing dataset of the generators.

Datasets can be pruned (to keep them manageable) if desired by selecting those images that are distinct from each other and that maximize the distance / entropy of the dataset. Special care must be taken to avoid using the synthetic data from the GANs within the generator dataset as this risks of introducing a priori bias.

Furthermore, expanding State-of-the-Art models and datasets from academia can be implemented (when applicable) to continuously extend the validation capabilities of Argus.

The largest challenge for the implementation of such a system is the legal one of how a third-party analyzes data obtained from another. Depending on the jurisdiction, it is likely that this implementation is complex if not downright impossible.

Additionally, convincing OEMs and other ADAS providers to grant third-party use of their data. Multiple examples of how this has been accomplished in the imaging sector can be seen in the success of Google Photos and Lensa, but hasn't been thoroughly explored in automotive scenarios.

7.5 Future Work

The scope of this solution is currently limited due to the timeframe for development. Further proposed improvements to the Argus product are as follows:

7.5.1 Thorough Research

A much more thorough literature research should be made to fully understand the aggregate conclusions in this field from academia and validate other alternatives and viable solutions. Multiple references should be analyzed and evaluated for the different pieces of the solution before a more definite implementation.

7.5.2 Robust and comprehensive delimitation

A more in depth delimitation of the solution is preferred. There are certainly many gaps that I haven't seen or thought about, or areas where my proposed solutions are lackluster, inexperienced or inefficient. A more thorough approach to this would require much more thorough discussions with multiple PMs, developers and researchers to ensure this is reviewed by multiple sets of eyes and a good solution can be iteratively achieved.

7.5.3 Evaluation extension

Multiple evaluation metrics can be extended in the generative model evaluation to provide a more robust and certain evaluation of the validness of the different models.

7.5.4 Better documentation and citations

Documentation should be extended with better descriptions, explanations, mathematical Latex equations, and multiple cites from the academic sources used in the system to ground the decisions being made and methods being used.

7.5.5 Code Implementation

The description of Argus should be implemented in a Proof-of-Concept working demo to prove that it works and start iterating based on actual use of the system. The implementation should follow the description in the architecture to make it robust and scalable from its onset, although a limited PoC can also be executed.

7.5.6 Dataset collection

Multiple OS datasets should be implemented to validate the potential of this idea and prove that the system works correctly.

7.5.7 Multimodal Sensors

The current version of Argus is limited to evaluating camera-based models that use images as their input information. Argus can be extended to use additional modes of information that are commonly used in ADAS systems. Different weather effects impact each of these sensors differently, so there are some that make sense to validate and others that won't for each sensor. For the adverse weather effects that do occur, the complexity of implementing this lies in obtaining the adequate datasets (with most being proprietary) that contain both clear weather and adverse weather sensor data that can then enable learning the domain mapping function of each.

7.5.8 CI & CD

Continuous Integration (CI) and Continuous Delivery (CD) via Jenkins or similar should also be implemented to ensure a smooth and flawless experience from the client's and ensure that the system is reliable, scalable and maintainable over time with faster development cycles. JIRA tracking and Github / Gitlab integrations should be implemented to

7.5.9 Scalable infrastructure

The current infrastructure of Argus is limited in its throughput capacity. A production-ready version should be implemented in scalable infrastructure that can easily add extra capacity based on demand and reduce it when unnecessary, to ensure optimal cost that fulfills all the system's needs. Load-balancing, resource allocation balance, and cluster-based computing instances can be possible solutions to help the system scale better based on client demand. The computing cost can also be automated by using Apache Spark, Hadoop, Flink, Dask, dCUDA, etc. Furthermore, code efficiency optimization would also be crucial to ensure the maximum possible efficiency.

7.5.10 Comparison vs Phenomenological Simulation

Further research needs to be invested in comparing the GAN-based weather simulation with the phenomenological simulation to truly evaluate if the GAN-based approach is adequately good at mapping and in an end-to-end solution to truly be similar in performance to a physics-based simulation, to ensure that there is actual empirical merit to using it as a better alternative.

CHAPTER

EIGHT

USE CASES

This chapter briefly explores the use cases of Argus and some examples.

8.1 Use Cases

Advanced Driver Assistance Systems (ADAS) are designed to enhance vehicle safety and convenience by providing various features such as automatic braking, lane departure warning, and adaptive cruise control. Due to the downstream impact of decision-making for safety systems, any ML model that uses camera-based inputs should be evaluated to ensure that they are accurate and reliable. Argus provides the safety and assurance to be highly certain these systems are safe to use in multiple environments and weather conditions.

Some example use cases are:

- Testing the robustness of an ADAS system: Argus can be used to test an ADAS system's performance under a range of different conditions, including different lighting conditions, weather conditions, and road surfaces. This can help to ensure that the system is reliable and effective in a variety of real-world scenarios.
- Dataset augmentation: Argus can be used to generate synthetic images that are highly real-like to improve the dataset bottleneck for multiple ML models used in ADAS systems.
- Verifying the accuracy of object detection: Argus can be used to ensure that an ADAS OD system is accurately detecting and identifying objects in the environment, such as pedestrians, vehicles, and traffic signs, even in adverse weather conditions.
- Evaluating the performance of lane detection: Argus can be used to assess the accuracy and reliability of a ML lane detection model, including its ability to identify lane markings and maintain a consistent position within the lane for scenarios where it is snowing or raining.
- Detecting and correcting errors: Argus can be used to identify and correct errors or biases in the ML model algorithms, helping to improve its accuracy and performance over time.
- Providing feedback for system development: Argus can provide valuable data and insights that can be used to improve the design and functionality of an ADAS system, enabling it to better meet the needs of users and enhance vehicle safety.

8.2 Client-Facing

Argus fulfills the following client goals in its client-facing functionality:

- Robustly validate and evaluate the performance of ML models at a scale with synthetically-augmented data.
- Have access to constantly-evolving, state-of-the-art generative models with zero friction.
- Visualize the evaluation of ML models, both qualitatively and quantitatively.
- Visualize the development and improvement of ML models over time to prove the newer versions are indeed optimal and observe trends.
- Be simple to use yet allow customization when required for fine-tuning.
- Quick aggregate identification of failure scenarios / conditions that require further work and focus.
- Complex / Edge case detection and monitoring over multiple model versions.
- Test dataset cleanup and sanitization.
- Model Explainability.
- Synthetic augmentation of Datasets to extend the training dataset with care and empower the best-performing real-world facing models.

The overarching goal of Argus for the client is to enable simple, yet thorough validation with a one-click or one-line-of-code that accomplishes this. This allows developers to fully focus on their algorithms while trusting that a thorough and complete validation always checks their model performance.

8.2.1 How we accomplish this

Each Argus client has access to their own client view, where they can

- Select or upload a proprietary Testing Dataset (and its ground truth) for use in the model evaluation.
- Select or upload a proprietary Dataset for use in the Synthetic image generation for dataset augmentation
- Select and use existing Sample Test Datasets to test out model evaluation and synthetic dataset augmentation.

Upon selection of a Dataset to use the system then proceeds to the synthetic augmentation selection. If the user just wants to validate, the default all-condition stack with the best performing model is always selected. Otherwise, if the user desires to configure the synthetic generation the system then prompts the user to select the desired adverse weather conditions from the catalogue (registered and enabled by the Argus team). For each adverse weather condition the system prompts a list of all generative models with its related KPIs for selection. Additionally, the aggregate of all adverse weather conditions is also listed. Selected weather stacks and model configurations can be saved to reuse in the future.

Upon confirmation, the dataset selected is thus synthetically augmented with the selected adverse weather effects. FDI and IS scores are obtained and presented to the user to evaluate the qualitatively the results from the GAN stack. If the goal of the user was to augment it for extending its training set, then the user can download those images. Alternatively, if the user wanted the synthetic test dataset augmentation to use it for testing of a model, the user is then prompted to select the model. It can be either a proprietary model or a sample model for testing.

Subsets of the testing dataset can be listed to allow detection of useful edge cases that are noteworthy, and allow comparing them across multiple models to quickly identify if a new version solves previously complex scenarios where the models struggled. Datasets should also be analyzed to ensure there is not oversampling of specific scenarios ensuring that the dataset is representative of the real world.

The model is then tested on the synthetically augmented dataset (including the real images) and the model results are visually presented. Insights are provided into the dataset. The model can also be compared with previous models to evaluate its performance over time, and view insights into specific cases that have improved.

8.3 Internal-Facing

Argus fulfills the following objectives in its internal-facing functionality:

- Allows for the development and deployment of generative models at a scale.
- Simplifies and minimizes the MLOps efforts required to ship the aforementioned generative models to the clients.
- Allows client user feedback as well as automated detection to improve the performance of generative models by identifying those scenarios where the image isn't sufficiently real-like
- Visualizes the evaluation of the Generative ML models, both qualitatively and quantitatively.
- Visualizes the development and improvement of the generative ML models over time, showcasing to the users that their synthetic augmentation is always state-of-the-art.
- Get insights into the performance of the generative models in a broader set of datasets (further enabling the performance showcase)
- Enables extending the generative training dataset for further improvements(data flywheel) with care.

The overarching goal of Argus for the internal team is to enable frictionless delivery of the best-performing models to the clients and evaluate those same models to constantly improve them and maximize their performance.

8.3.1 How we accomplish this

Each Argus team member has access to the internal view, where they can

- Add or edit Adverse Weather conditions from the catalogue.
- Add or edit Generative models per-condition or full-stack.
- Edit and upload internal Datasets (and their ground truth) for use in the generative model training, validation and testing.
- Edit and upload Sample Datasets (and their ground truth) for use in testing.
- Evaluate Generative models with internal Test Datasets and evaluate their results for internal comparisons with previous ones and performance showcases.
- Identify scenarios or conditions where the generative models are underperforming to enable focusing on those for further improvements.
- Extend the internal datasets when applicable.

CHAPTER

NINE

SAAS

This chapter briefly explores the economics of Argus as a SaaS.

9.1 Income Sources

Argus is meant to be a for-profit endeavor, so it must be able to generate income to be sustainable. As a SaaS the primary source of income is recurrent due to monthly billing. This is sustainable long-term as it ensures that a continuous development can be run with a positive cash flow. This ensures that generative models are constantly being maintained and upgraded, ensuring clients always have the best-performing state-of-the-art models (and thus the product is always kept valuable). As the generative models are of use to multiple clients, a small team can efficiently deliver noteworthy improvements to a much higher developer codebase, freeing their The main income sources are:

- Monthly team-wide subscriptions according to the project needs.
- Per-shot evaluation of models outside that go over the subscription budget.
- Customized solutions or hosting for clients that require more complex strategies.

The following are some financial advantages of using the Argus platform for development

9.1.1 Developer costs

For clients, Argus makes economical sense by reducing the time spent by developers in developing and maintaining evaluation models and frees that time so that it can be spent developing the actual ADAS ML models. An In-house development of similar pipelines internally within each company would require developers to constantly develop new generative algorithms, collect datasets and evaluate them to obtain their best performing models, while also maintaining DevOps and MLOps operations (and incurring in their costs). Additionally, as multiple clients can benefit from the same model, this allows each client to have a proportionally reduced investment. By offloading the development and dataset collection cost to Argus, they can reap the rewards with minimal investment. Thus, this arrangement is beneficial for all parties and is economically sound. As Argus is fully focused in evaluation and generative models it achieves specialization efficiency, thus enabling it to efficiently develop evaluation pipelines and datasets.

9.1.2 Dataset costs

Dataset collection for ADAS systems poses a significant challenge. A fleet must be assembled and tested by actual drivers, additionally some weather conditions need to be met for it to be tested under those circumstances (which can depend on pure luck). By allowing the creation of synthetically augmented data to expand your dataset from clear images, Argus empowers data collection efforts and reduces the cost and time necessary to obtain statistically significant datasets.

9.1.3 Testing costs

Evaluating ML models in the real world is quite expensive especially at a large scale and when considering complex scenarios. Testing needs to be done with a fleet of actual drivers, and it can be necessary to test under particular adverse weather conditions. This testing can potentially occur with every new model release if the acquired data is insufficient to evaluate correctly the data with statistical certainty. Argus mitigates this by allowing synthetically-augmented testing at a scale, ensuring the dataset is as descriptive and representative as required, and potentially extending clear/overcast tests with other conditions. Testing is done with minimal effort and overhead, and validates the models directly. By providing insights and analytics of the results, Argus further reduces the development cycle turnaround time.

Additionally, rework due to testing failing on the real world due to the Sim-to-Real gap is thus expensive as it requires this setup to be performed again. This is increasingly more expensive if there is a need for a product recall caused by a lack of testing on adverse weather conditions (Tesla has already recalled vehicles). Argus helps detect models which might not work in the real world beforehand, preventing faulty models from getting delivered to the vehicle.

9.2 Costs

The main operating costs of the proposed Argus system are the following:

- Developer / Engineer and other HR salaries
- Data storage costs for datasets (S3, GCS, Azure Blob Storage, ClearML Hyper-Datasets)
- Infrastructure DevOps costs (Self-hosted servers, Cloud-based servers)
- GPU training costs (GPUs, AWS, GCP, Azure, ClearML)
- Experiment Analytics and Logging costs (Self-hosted, Neptune, Tensorboard, Weights and Biases, ClearML Experiment)
- Metadata storage costs (Self-hosted, Neptune, MLFlow)
- Model Deployment costs (Self-hosted, Seldon, ClearML Deploy)
- Model Registry costs (Self-hosted, Neptune, MLFlow, ClearML)

Most of these costs are highly variable, and tend to be more expensive when the system is in production and under client load. The costs of the system are highly dependent on the number of models that are being trained, and the number of datasets that are being used. Most of these systems are billed based on use, which allows flexibility when scaling up and down the system. A correct budgeting and optimization of usage costs can ensure the optimal usage of each component to make the system as cost-effective as possible.

These costs tend to be highly beneficial as any system that empowers internal developers to be more efficient has a direct impact on the bottom line of the company by improving efficiency and availability. For example, a MLOps system that allows for the training of models to be automated and streamlined can free up the time of the data scientists to work on other tasks, and can also allow for the training of more models, which in turn can increase the productivity of the company. Similarly, analyzing and visualizing efficiently the results of generative models can increase the turnaround time of the development cycle, and thus allow faster releases to satisfy the needs of the clients.

9.3 Licensing and Legal

The legal and licensing topics are of crucial importance for the development of Argus and must be thoroughly analyzed. Legal implications can be found in the following topics:

- Proprietary dataset collection
- Client dataset usage
- Open Source dataset usage
- Open Source model usage
- Model evaluation

In terms of Open Source dataset, as previously mentioned the licensing of the dataset must be taken into account. Most OS datasets won't be able to be used within the Argus system as it is a for-profit endeavor. Open source evaluation models might be restricted to the same limitations.

Proprietary dataset collection must be taken within the limits of the legal framework of the country that is desired to be used by the fleet of vehicles.

Client datasets when applicable must be used and integrated only when the legal framework allows it, as it must be very clear that the data can be handled by a third-party and is adequately licensed.

Particular care must be taken when evaluating systems that can have trouble with obtaining the representative data due to legal reasons (i.e. evaluating private garages), as there would be a need for the dataset to be representative to have a usefulness in its evaluation and the legal limitations might restrict obtaining enough data to be representative.