

Subscribe to DeepL Pro to edit this document.  
Visit [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document)for more information.

**Slide 1**

What are MLOPs? It's everything that will help and support the fact of putting artificial intelligence algorithms into production, they are not tools or technologies; it's mainly a question of culture and good practice.

My name is Guillaume Chervet and I'm a ML engineer at Axa France. I don't know exactly what that means, but basically I define my role as accompanying data scientists and putting artificial intelligence algorithms into production.

I work in a team that mainly works on real-time automatic document reading projects. Usually this will mainly involve one or more deep learning algorithms.

Today, almost 95% of projects fail at the production stage. This figure is starting to fall. This does not surprise me because on this type of project, which is at the top of the line in terms of complexity in all areas, all the keys are there for it to fail.

**Slide 2: intro trio**

The first problem is a human problem, so today most applications have devs and ops and it's already not easy to get everyone to work in a united team. Now we're going to add even more specialities to the data professions. Data experts.

Today, it is almost impossible to be an expert in all three areas. This type of project is top of the line in terms of complexity in all areas of technical expertise. It therefore requires accepting the constraints of others and trusting them.

**Slide 3: expensive intro**

Add to that the fact that there are more actors: we are dealing with projects that are generally extremely expensive.

So this is an email classification project that has worked very well. For us it's a bit of an example project. Today the cost of running the project per month is about €9,000, which makes the project cost a little over €100,000 per year. I think we can still divide the costs by 2. Today on this project we classify about 10,000 emails per day, which makes the cost of classifying an email at 3 cents. If, for example, we carry out one re-training per year, for example, we will have project costs which mean that today the cost of classifying an email will be around 4 cents and here again I am not talking about the implementation of the project which cost a little more than 1,000,000 euros.

**Slide 4: expensive intro**

Here is a functional example of mail classification.

For information, all the images you see where there is a small text underneath in python style, are images generated with artificial intelligence algorithms.

**Slide 5: Contents**

As you can see in these projects at the top of the technical complexity, all the keys are there for it to fail.

This presentation is divided into 4 parts.

In the first part we will introduce Deep learning and its issues.

In part 2nde we will explain what we expect from a project in production at AXA France.

Then in the third part we will have fun making a recipe. A recipe of what to do to make a project fail. The idea is that we have had experiences that help us to improve today.

So, in Part 4e we have learned from these experiences that will enable us to be on the path to making a project work.

**Slide 7**

One subset of artificial intelligence is machine learning and one subset of machine learning is deep learning.

**Slide 8**

You will often hear of structured data usually being represented by an Excel file. In general for this type of problem where there is a limited number of inputs and outputs. We will use classic machine learning algorithms.

**Slide 9**

When we have data such as sound, image, video, the number of possible output inputs is generally much higher. As a result, we will generally use deep learning algorithms that use a more complex neural network.

**Slide 10**

So this is a diagram that came out of the book by François Chollet, who is the creator of Keras, which I really advise you to read.

When we do classical code, we will usually have rules plus input data and we will write an algorithm that will allow us to generate answers.

In machine learning it's a bit different, we're going to need input data and also a lot of answers. Not just a little, but a lot, thousands of examples, 10,000, 100,000 or even more, which will enable us to write code that will generate rules: an artificial intelligence, technically called a "model".

**Slide 11**

In production the classic code or ia is used in the same way.

We receive an input, we execute, and then we get the result back.

**Slide 12**

In the past, we used to talk about a project: a centric model, because we were not sure of being able to make an adapted ia. Today, most of the points on the ia side: at least for our needs at AXA France, already have viable solutions. We know that we will be able to create an AI for our use cases. However, the heart of the problem, really the thing that makes a project feasible or not, is the data. If you have good data, good quality data, lots of inputs and answers, that's really the key to the success of a project. Data is your treasure that will make the difference.

**Slide13**

So now we're going to explain how a project works. If we take our email classification project and want to start the project. The first thing we need to do is to get some data: entries and replies. On this project, it's a good thing we already had emails and replies because in production we have humans who classify emails manually. We already have the data and the responses, so it's perfect to build a data set.

**Slide14 and 15 and 16**

Then with this data we can Iterate to create an AI and build an API that we can put into production. Our application in production will be able to be consumed by client applications.

**Slide 17**

In production it is extremely important to set up what is called a feedback loop. A virtuous loop. In the event of a classification error, the client application must allow the error to be corrected manually. Over time, we will be able to build up new data sets of better quality and thus improve our artificial intelligence.

**Slide 18 Demo intro**

We're going to do a little demo of how a project works based on a well-known Kaggle cat and dog dataset.

**Slide 19 Demo Data**

Before starting the small demo. Let's imagine that we have 20,000 files available. We'll use 17,000 files to train the model. Then we'll keep 3000 files aside for validation, to test our model on data it has never seen. This validation data will also be used to test the production environments.

**Demo Time**

**Slide 20 After Demo Datadrift**

If we feed our models different data, data they have learned from. The predictions may deteriorate.

For our email classification project, for example, if the car report changes template, the AI may not recognise car claims as well. In this case, the model must be re-trained and re-delivered to production.

Let's go back to the demo with our cats and dogs. Now let's imagine that our users start to use dog and cat plush pictures which was not planned at the beginning, the predictions don't work correctly anymore. Another example with drawings of dogs and cats.

**Slide 21 The Grail**

The Holy Grail for us would be to be able to re-deliver to production with almost no manual action

**Slide 22 100% code**

This is a Google slide, a Deep learning project is a little less than 5% data science, but it is 100% code!

**Slide 23: Contents 2ème part**

Now I'm going to explain to you what we expect from a project in production at AXA France

**Slide 24: Needs**

The most important thing in a project is to meet the user's needs and in general what the user wants.

He wants good prediction quality, so that production is not expensive and fast. At AXA France, we set ourselves response time constraints that must be less than 10 seconds for the automatic reading of documents.

Safety is non-negotiable.

We want the projects to be monitored. The idea is that if we have a data drift, it could lead to a production incident. If we're on a major project that can block the box, we're not going to wait for the incident and block the box for weeks.

The last point is that we must be able to re-train and redeploy quickly. If we have a major incident that blocks the box, we have to be able to release it as quickly as possible and not wait months to unblock the situation.

**Slide 25 Functional diagram**

So this is the workflow block diagram that explains how to read a French driving licence. We also read old French driving licences, but to simplify, we don't show it here. The idea is that we will receive an image or 1 PDF as input and we will first perform a first step where we will split this PDF into one or more pages. On each page we will run the same zoning algorithm to detect the front and back sides. After that we will enter specific algorithms for the front and back. Let's take the front side at the bottom, we will first put the front side image back to the right with a front side specific algorithm. Then execute another algorithm to zone the fields specific to this front side and finally execute an OCR to finely extract the text from the fields.

If we were to host an API that exposes this service in monolithic mode. We would have several problems, the first is the response time, i.e. when we call this API in monolithic mode, what will happen is that each algorithm will consume all the CPU resources of the machine. The consequence is that the pipeline will run sequentially, the response time is greater than 30 seconds. Another problem is that during these 30 seconds: if another call arrives at the API, this call will have to wait for the end of the first processing to start processing the new processing. So you'll tell me that we can add more machines, but another problem is that if we host this in monolithic mode, we need machines with about 32 gigabytes of RAM, which is very expensive and which will take a long time to start the application. It's going to be very complicated to be able to scale quickly and adapt to the user load.

The solution is to set up microservices, we'll host each algorithm on different resources which will allow us to finely configure the RAM and CPU needed. We will be able to run the processes in parallel and scale quickly.

**Slide 26: Function**

How does it work technically? We use what are called functions. Each algorithm is hosted on a function. This function listens to an input queue and as soon as it has free time, it will be able to unpack the messages that arrive in the queue and process them.

For example let's take the SPLITTER which extracts the pages of a PDF, the function will get a message from the Queue, the files are not stored in the Queue but they are stored in a REDIS we will receive only the identifier of the file in input; we will fetch the image in REDIS to execute our algorithms and for example if we have a PDF with 2 pages we will extract the 2 pages then generate 2 identifiers; push the images in REDIS with the identifier and finally add 2 messages in the downstream QUEUE to be able to call the following service, in this case the Permit Detection Zone.

**Slide 27: Scale**

The advantage of working this way is that we can scroll quickly and support the load spikes. Imagine that all of a sudden we will receive 500 permits, the number of functions will increase to be able to unpack the messages faster. There is also a financial advantage because you only consume what you need.

**Slide 28: Architecture**

So that's the architecture diagram. If you find it complicated, that's normal because it's complicated.

I'm not going to describe the whole scheme but, in summary, we use an asynchronous architecture. The first POD will receive the file and push it directly into redis, then with the file ID add a message to the QUEUE upstream of the splitter. Then the splitter will retrieve the file and process it, etc. etc.

This architecture works for real time and also in batch mode.

**Slide 29: Difficulties**

**These projects are not without their challenges.**

I will present the 2 most important ones

**Slide 30: Triptic**

First of all, there is a real triptych of choices. A cursor to be positioned between the quality of prediction, the response time and the cost of infrastructure in production.

This means that if you want to have the best prediction quality you can accept to pay more or lose response time.

On the other hand, if you want to be able to respond more quickly, you have to be willing to pay more or lose prediction quality.

And conversely, if you want the project to cost less in production, you must either accept to increase the processing time of the answers or to decrease the quality of the prediction.

**Slide 31: Finding the differences?**

So do you see the differences? The differences in these images we can't see, but the AI can! I first experienced this problem on our very first project I worked on. As a good developer, there was a Pillow library that is used to resize images that was out of date. As one must always maintain libraries. I updated it. What happened next was that the predictions were almost not working. So I looked at the images. Identical images to my eyes, impossible to understand. And in fact, the peculiarity is that this is really a problem for the AI, which no longer recognises the images and is sensitive to the evolution of the algorithms. The consequence of this problem is that if in production we don't use the same versions of python, nor the same versions of each library and not even the same OS, we are almost sure that the production will not work as expected.

Let's go back to the demo. If we go back to my cats and dogs, the AI algorithm used here receives input from images resized to 200\*200 pixels via the Pillow library. It has also been trained with input images resized with Pillow to 200\*200 pixels.

On the driving licence project, we replaced the image resizing with the OpenCV library, which allowed us to save a little less than 2 total seconds on the complete processing of the pipeline.

In this demo I have also implemented a version where the images here in production are resized with OpenCV before being given to the AI model.

Therefore, what happens is that if I make predictions with resizing via Pillow and then with resizing via OpenCV I don't get the same results.

I did a test with 12500 images and out of 12500 images you have about 500 images that don't predict the same thing.

**Slide 32: Sliding impact**

That's only a 3% difference in prediction, but if you take a complex pipeline you're going to be chaining algorithms with different AIs. What happens is that the 3% will propagate all along the pipeline, so that in the end the prediction rate becomes very low.

**Slide 33: Summary part 3**

So, the idea is that now for this part we are going to do a little feedback to explain our experiences that have allowed us to learn what not to do. To do it in a funny way, we're going to open a pizzeria and we're going to organise ourselves to serve pizzas to our customers. We're going to set up everything we shouldn't in order to make our pizzeria run smoothly.

**Slide 34: Elisabeth**

To begin with, we will organise our team. In a project what happens is that there are generally three main phases. In the first phase of exploration we will test if the project is feasible, then we move on to an industrialisation phase and finally a Deployment and Run phase.

I'd like to introduce Elisabeth, who is a data scientist and who will be working alone on the exploration phase.

**Slide 35: Elisabeth**

So Elisabeth is an excellent cook, we'll ask her to knead the pizza dough and to roll it out well, and she does that with all her heart and she does it very well.

**Slide 36: Elisabeth**

There, now Elisabeth has made her pizza dough and rolled it out.

**Slide 37: Elisabeth - Hicham**

This is Hicham, his role is to distribute the ingredients on the pizza dough.

**Slide 38: Elisabeth - Hicham Hop**

Now Elisabeth is handing over the pizza dough and ingredients to Hicham. As you can see on the left, Elisabeth forgot to give the ham to Hicham.

**Slide 39: Hicham**

So now it's Hicham's turn to play and it's his turn to spread the ingredients on the pizza dough, but he realises that the ham has been forgotten and he also thought that Elisabeth would have pre-cut the ingredients. Unfortunately, he does not have a knife at his disposal.

Hicham, as a good developer, does everything he can. He takes the ingredients and spreads them as best he can on the pizza dough.

**Slide 40: Hicham**

See

**Slide 41: Hicham - Khalid**

This is Khalid; he is our Ops person responsible for baking the pizza and delivering it to the customers and it's his turn to play.

**Slide 42: Hicham - Khalid Hop**

Hicham hands over the pizza with its heart to Khalid.

**Slide 43: Khalid**

Khalid is left with a well-stuffed pizza, but someone forgot to give him the instructions. He doesn't know how to cook it.

**Slide 44: Khalid**

So Khalid does what he can to cook the pizza, but damn it's a special pizza, he didn't know that and now it's burnt.

**Slide 45: Team summary**

To sum up, here we have organised our kitchen in such a way that we separate the roles and do not have them working together and at the same time in "teams" Elisabeth, Hicham and Khalid. If you want to make your project fail, this is the right way to organise yourself.

It is important to know that the very choice of libraries at the beginning of the project has a direct impact on production, so not working together, not caring about production at the beginning of the project, is one of the keys to failure.

**Slide 46: Lilian**

I almost forgot to mention Lilian. Lilian came this morning to deliver salt, which is one of the raw materials for our pizzas; but he made a mistake and delivered sugar! Our pizza will be good.

**Slide 47: Table**

Now we're going to set the table for our clients. So if you want to make a project fail it can be a very good idea not to focus on the user need.

First example, on the driving licence project, we were asked to read the licences in less than 10 seconds. When we started working on it and took the very first algorithm, the PDF Splitter, and put it into production. We quickly realised that on the production environment, the extraction times for PDF documents were taking around 15 to 30 seconds which was problematic. The data scientists didn't realise this because they work on very powerful machines that perform the operation very quickly.

Second example, when we were a little further along in the project. We realised that some readings were very slow and were clogging up the systems. When we looked a little more closely at the permits that were not passing. We realised that these permits, in general, were almost illegible. For this kind of case, it would have been a good idea to go to the users and ask them if it would not be better to immediately show in 2 seconds that the permit is not clean enough and not readable rather than trying to read it "no matter what" and that in the end, after 40 seconds, almost no field is read.

**Slide 48: Aperitif.**

Aperitif

So now we're finally going to serve our first clients a drink. When you start a project, especially if you want to do agile and have a fast Time To Market. It can be a good idea to always test all the classic code approaches. Rather than immediately doing Deep learning.

Setting up a Deep Learning algorithm is a bit of a gas factory, it's complicated, you need a lot of data and it costs a lot of money.

First of all, classic code is faster to implement. It allows for rapid iteration and in general the processing time is faster and requires less CPU resources.

However, classical code will not allow me to be as good at prediction as Deep learning. Deep learning will solve very complex problems much better, but the iterations are long.

Generally speaking, if you start with classic code and you don't manage to achieve the desired results, it's not a waste of time, because this classic code will allow you to save time on the annotation part. You can pre-annotate your annotations and also you will have a fallback system in case the AI does not predict well.

**Slide 49 Aperitif 1**

So this is an example we had on the driving licence project. To straighten a front or back side, we used two AIs each time to put the image back on the right side. The problem was that these algorithms were very resource-intensive and therefore very expensive and took too much processing time. We were able to replace them with more classic algorithms with the same quality of result, the consequence: a great saving in response time and money.

**Slide 50 Aperitif 2**

We had the same problem with the email classification project. What was happening was that we had a deep learning algorithm that had to put the documents right. It wasn't doing it well and it was taking even longer than reading the document itself. So we insisted on looking for algorithms and we managed to get almost all the images perfectly straight without AI, extremely quickly in less than 0.3 seconds and with very little resource consumption.

And who knows, if tomorrow we want to improve reading performance even more and we have the budget, we could train an AI.

**Slide 51 Aperitif 3**

Last example for our aperitif, we tried to replace the custom written Deep Learning algorithms to detect the fields to be read by free algorithms found on the internet already done. We found that the prediction quality was almost identical. The only difference was that this algorithm was not as good at detecting text on permits that are really completely unreadable. If we had excluded these illegible permits from the beginning, we could have used this algorithm and thus saved time to market and made a lot of money.

**Slide 52: Salad**

Now we've finished the aperitif, let's move on to the starter. It might be a good idea not to industrialise your training code. If you don't industrialise your training code. When you have to start the production phase, you are almost obliged to re-train the models because you are almost obliged to adapt the code. If you haven't industrialised the training code, i.e. if you've stayed on Jupyter Notebook without versioning the data or the code. It takes almost the entire project time to re-train the models.

So now we can move on to the main course and serve our pizza to our customers

**Slide 53: Spaghetti**

In fact no, we made a mistake, it's not a pizza; we served a plate of spaghetti. Code quality, unit testing today is extremely important for project maintenance. Don't do it if you don't want your project to succeed.

For example, on the email classification project, we had a phase of extracting data from a lot of emails that took about 2 weeks of processing time. There were two possible choices, wait 2 weeks to validate a code modification, or take 20 minutes to perform a unit test. With this example, the Data Scientists understood the value of unit testing. Thanks to this we were able to iterate very quickly and our processing time is less than half a day.

**Slide 54: Code without testing**

You see this example of code. This is classic code that you can usually see in Python. Without a unit test, without an example of inputs and outputs, I am personally unable to know what this function does.

**Slide 55: Salt**

A little pinch of salt on our pizza.

If you really want to make your project fail, it can also be a good idea to work on separate GITs. A single GIT is simple and pragmatic, it allows you to work in team mode via PullRequest and focus on production. It minimises the number of manual actions and thus simplifies automation. So if you don't want to work in a team and make your already complex project more complex, separating GIT is a good recipe for failure.

Separating the GITs is also great for having code mismatches between training and production and therefore not working.

**Slide 56: Dessert**

The icing on the cake for dessert!

Not monitoring your production, that's easy you have a major project, you don't monitor it. The model starts predicting anything and you are not aware of it. You are heading for a major incident that will block your entire company and potentially sink it.

**Slide 57: Coffee**

Small coffee shop

It may be a good idea to focus on statistics alone. Statistics are a good indicator. However, in my experience there is nothing like looking at and visualising the data. With data, you can see exactly what is happening in a complex pipeline extremely quickly. When there is a problem, you always have to go back to the heart of the matter: the data.

**Slide 58 Addition**

Let's go to the check

We shouldn't wait for someone to come to us to start paying attention if the project is too expensive. The planet has limited resources, so we might as well start thinking about using as few resources as possible and being as efficient as possible.

If your project is too expensive, it will not live long.

**Slide 59: Contents 4ème part**

With these experiences we have learned and now we can try to put ourselves on the right path and put all the chances of success on our side.

**Slide 60: Experimentation**

We will start again from the exploration phase. What is important from the start in the organisation of the project is to get all the players to work together with a single objective: that it works in production.

Humanly, to understand that the problems of some are the problems of others so that it works. This is really the heart of the success of a project or not. This is also why the title of the presentation is called ML OPS is a human adventure, because it is really a culture of sharing associated with the fact of trusting each other.

This exploration phase will help to assess whether the project is feasible.

**Slide 61: Experimentation Set up**

At AXA France we work directly with production data for AI issues. To secure this, we will ask Khalid, our Ops, to deploy a development environment. Only a few people will have access to this development environment, which is on AzureML.

**Slide 62: Experimentation Elisabeth and Hicham**

From the beginning Elisabeth and Hicham will work together from GIT and a DataLake. It will be mainly Elisabeth who will be active in the first phase but accompanied by Hicham who knows the production issues.

**Slide 63: Experimentation Annotation**

Quite soon we will have to start annotation phases. We're going to meet up with Lilian. It is very important to work in collaboration with the team that will be annotating the data. Why is that? It is really the data that is the treasure of your box.

What we're going to do is to first make small batches of data. This will be used to refine the annotation rules. It is extremely important that the annotations are carried out in the same way by everyone.

Imagine that you have to Zoner cats. Imagine that you have a cat on a table and its tail goes under the table cut off in the photo and becomes visible again on the other side. One annotator may zoner the cat without the tail at the other end of the table and another annotator may select the cat with the two pieces of tail included.

With its data, AI will not be able to learn well, it is very important to work in a team and to refine the annotation rules according to each specific case we encounter.

Once the rules are well defined, the volume of data to be annotated can be increased.

**Slide 64: Experimental model**

Once we have annotated the data, we can train a model and then, to automate and save time, we will set up a continuous integration CI which will allow us to replay the training automatically in an almost industrial way. The deliverable is not the model. It's the code that's in GIT plus the data that's stored/versified in the DataLake.

**Slide 65: Experimentation Deliverables**

You can see that here we already have a well spread pizza dough with all our ingredients and a knife ready to use, so we really have everything we need.

At the end of this step we should all be able to estimate the metrics.

* That is, the quality of prediction
* Estimated cost of production infrastructure
* Estimated response time
* We already have a versioned quasi-industrial training code and the versioned data is our deliverable

**Slide 66: Industrialisation**

We will be able to enter the industrialisation phase where the aim will be to develop and bring into production APIs that host the models.

**Slide 68: Docker industrialisation**

We're going to set up CI continuous integration which will allow us to build docker images that will retrieve models from the model registry and then build our APIs with inference code and push them into a docker registry.

**Slide 69: GitOps industrialisation**

At AXA France, we deploy our docker images on open shift. We will ask our Obs to build the development and acceptance environments for us. We deliver using GITOPS, i.e. we will have 4 branches in git, one for the development environment, one for the recipe environment, and 2 others for pre-production and production.

This will allow us to deploy our applications via GIT handling.

**Slide 70 GitOps industrialisation ml-cli**

Today we have tools that allow us to validate the proper functioning of the application in the environments. We check that we have the same quality of prediction as in training. We also check the response time and we finely configure the infrastructure sizing.

Once the development environment is validated. We can deploy in recipe

To do this, we perform a PullRequest on the recipe branch.

**Slide 71: AzureML Production Industrialisation**

At this point we will enter another phase of the project. We will ask our Ops to create a new AzureML training environment still on production. On this environment we will not have write access, we will only have read-only access.

**Slide 72: Big** CI **industrialisation**

We will set up a pipeline that will allow us to re-train the whole chain automatically to be sure that we have no phase shift at each stage and rebuild each model in order, then build all the docker images that we save in a docker registry.

**Slide 73: GitOps Dev Industrialisation**

We can now, via GIT manipulations, deploy on the development environment and validate the environment.

**Slide 74 Industrialisation GitOps Rec**

Deploy in recipe and validate the recipe.

**Slide 75 Industrialisation Ready**

At this point you already have an almost pre-baked pizza.

**Slide 76 Industrialisation GitOps Pre-production**

For safety reasons, we do not have the right to deliver in pre-production. Only the OPS can do that and they will deliver this pre-cooked pizza in pre-production. We validate the environment.

**Slide 77 Industrialisation GitOps Production**

Then Khalid deploys in production.

**Slide 78: Deployment & run**

The API and templates are deployed and now the live application is needed.

**Slide79: Deployment & run Monitoring**

For the moment we have set up functional monitoring to monitor the DataDrift using Prometheus and grafana in addition to the technical monitoring.

This monitoring allows us to easily link the metrics to automatic alerts that are sent to another team specialised in production monitoring.

Then, we work on the implementation of the feedback loop and we will connect to this feedback loop to set up much more detailed metrics and alerting.

**Slide 80: End**

Today we have 2 ways to share code in the same GIT.

So today we position a directory called production. As in Python everything is a module, the training code uses this module called production to perform the inference for the scoring.

**Slide 81: End**

Another technique we are testing to share code and work together is to make local packages that we reuse locally.

**Slide 82: Open source**

MLOPS is not tool-centric, however through our projects we have been able to implement tools and open source them. It is a pleasure to share them with you.

The daily clean allows to automatically shut down Kubernetes environments.   
Ecotag enables annotation and GDPR compliance.   
mlc-li is our integration testing tool.

**Slide 83: Conclusion**

MLOPS is above all a culture of sharing. Today we have projects that are at the top of the line in terms of complexity and if we don't put everything in place so that the experts all work together and understand that the problem of one is the problem of the other. This cannot work. The key to success comes in this sharing and working together.