Q1. Could you describe your process for developing an ML system?

Yes, for me, programming actually comes relatively late, so we first talk many loops with the customer about what his pain points actually are, because he can't articulate them in data science language. And then we look to see if this is a data science or AI problem at all, whatever you want to call it, because it's often not, because our customers who come to us don't have a technical understanding of it, of course, so you have to pass it on. If we then also came to the conclusion that this is a problem that needs Data Science, that you could solve with Data Science, if it's a trivial problem, which it usually isn't, then I would have tacked directly from my own wealth of experience, mostly then with Jupyter Notebooks and Visual Studio Code try out the common libraries for my problem, see how far I get, if it's a little bit something, If it's something I haven't done before or if I think it's something where we're not yet in a typical product-ready environment, but rather something that's still state of the art, then I would have done a literature search and just looked at where things are going, where the current state is, what the best solutions are for the problem in question, and then it just depends on what the problem is and what literature I can find. I also look at a bit more advanced literature. The experimental things, somehow quite fancy new architectures that haven't been tried before but now maybe the first or second paper is out, I skip all those, so if I don't even find an approximately working library for it, but it's just a paper that I can't reproduce anymore, then I don't spend any time at all. About JN and Visual Studio Code: Well, I don't use this web interface again and mostly just for a mixture of documentation and testing, then I usually have two notebooks if I don't want to try out several things, where I first write down in between what I want to produce at all, then I usually have a JN for visualization. And use that quasi as a complete documentation, which I can then also give to my colleagues and say that is what I have tried out, these frameworks, these are my motivation, reasons etc are simply that I have no media break in this phase, because of course I throw away a lot or the many notebooks have then also served their purpose. It doesn't work, it's not a promising approach or whatever, then it just disappears in the documentation drawer and then it doesn't show up again for the time being. When I first find a few usable results that I can talk through with the customer, I usually do that. Depending on how much time has passed, this step then also gives a bit of a forecast of what I think is feasible in the future, what our initial research has shown. And yes, depending on that, it then starts. One phase, the company still describes it as prototyping, but I would actually already describe it as product development, so then the next, so prototyping goes on for several years, so under a year a prototype is actually almost never finished and for me a prototype is what I make in the first month or so. And everything else then not and then my approach changes drastically. During prototyping I don't automate anything yet for which I would write any scripts now, build any CI/CD pipeline. Because I also throw away so much simply. If the problem is already so far marked out that I know I move at least only in a solution circle, then I begin already times to consider me a little bit some Design Patterns and to build a few simple classes, thus I control then however also again with JN, but simply around there a little bit reusability at the beginning to ensure, but I do really only if I can assume that I understand the problem already well and the problem does not change again 10 times, which is in most cases also still there. I've had the luxury in recent years that the data engineering hasn't been that complex. Then it's mostly about building things incrementally, so I'm actually always a big fan of having a breakthrough as early as possible and having a minimal solution, kind of getting it up and running and not actually building up any technical debt from the beginning and automating everything right now. So we usually have two instances in the first place, a production instance and a test instance, and then also

work directly with Docker and make the automation process as agile as possible or in such a way that you can make changes as often as possible without being punished for mistakes. A code repository actually contains everything that is not raw data, but everything around it. Actually the complete source code. For this, you open the JN on the side and then you copy over all the stuff that you have programmed before, embed it in design patterns that you would like to have. So it's a lot of copy-pasting and then developing around that a runnable solution that works well with the other components. Most of the time it's then a microservice architecture. Actually, a JN is usually split into 3 services, so then of course you try to separate the data wrangling from the ML components and feature engineering if you want it to scale at some point. Until one then the old state of the JN into the pattern has transferred, remains then always so simply left besides on and one copies oneself the things over until it runs. The JNs are also in the repository. We set up the ML pipelines ourselves. Mostly the microservices nudge each other and then talk to the data processing. This is done via shell scripts and then the containers settle it among themselves as to who does what. In projects I'm involved in, I live test-driven-development as best I can and am an absolute advocate for it otherwise you don't get anywhere with changes. I try to learn new things and live them as much as I can.

Q2. Do you continue experiments after deploying the ML system? If so, how do you go about doing this?

Sure, as soon as something new explorative comes up, we always do it in JN and if it's based on something old, then we usually continue it there. Unless, of course, there is a further development of something that we have already committed to, then we do that in the containers as well. So then we build the necessities for it. So if we want to do any evaluations, hyperparameter optimization, etc., then we build that into the containers, then we don't do it via the JN. We actually only go to JN when we want to add something completely new, so before we want to send it to production. Then it's either a success or non-success this exploratory spike. And then there are really only two ways - either it was a success and we use it or if not, we don't. Then the JN is ticked off. If it's a success, then it ends up in a backlog with a request to integrate the feature. Sometimes there's a session together where they figure out how we can best fit it into the existing system, which at that point is hopefully modular enough to handle changes without us having to rip apart the entire code base. And then somebody takes it upon themselves to build that pipeline step into our previous architecture. Most of the time there's a component that takes control of other components and then you want to say Ok, now we have 9 containers and at this point please insert that into our ML pipeline. Therefore, depending on whether it is the first model, you can also go faster, there is then show value in focus, there is then also no somehow automatic, somehow shadow deployments or so what would then somehow allow. Testing the model against others is then the first, the second throw, then yes the one with which you also simply show the value and the processes then come downstream when the value is provided. Often the notebook is then thrown away. However throw away now also not so so it is already still comprehensible for everyone, but often it is just redundant, because the function then simply already neatly in the utils are stored and there then also simply to be pulled. I usually have an exploration folder in the whole structure, where they are also left in, but even that is now cleaned up more by me, also simply in view of the fact that someone else goes to it and at some point comes the point where someone else goes to it, even if it's just me. Hm, so it is yes it depends on how you also use it in productions. So then we will just have the model artifact. Then in the vast majority of production cases we

also have a model registry or we have it in some file storage from which we then somehow fetch it.

Q3. Which tools do you use?

"already answered in Q1. [Interviewer asks if plugins are used] Actually pretty basic. Only if I now and then comes, on what you just do not need to use, so if I rely on Google Collabs or something, then use any additions to work with the file system.
Otherwise a plugin so that the libraries always reload automatically without me having to restart a kernel.
Sometimes some additional stuff for rendering. Notebooks sometimes have some audio problems or when rendering a video, it often doesn't work. But that is always specific to the framework that is used, mostly you load some custom solutions in there."

[Q4 skipped due to time constraints]

Q5. Which issues do you encounter when working with Jupyter Notebooks?

We really only have a small spike with JN and then we leave that as well, so we don't have the expectations of that. It has to be reproducible. If it worked with our services and then after a month some dependencies have changed and the JN doesn't work anymore, we've had that many times, then it has to be small enough that you can rewrite it