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

"Yes, exactly, so perhaps a very brief word about <company> in advance, that is embedded a bit. We are asked above all for projects where the clear goal is actually there relatively early on, that you don't just do a feasibility study, that you consciously have the goal in mind of getting things into production very quickly. In other words, it should definitely not end up as a POC in the customer's drawer. Sure, it can come out that is simply not feasible, the idea that the customer would like to implement, then it is just that, then does not remain in the drawer but in the trash. But actually, the declared goal from the outset is to get things into production promptly, and accordingly, we actually work very iteratively from the very beginning, very closely with the customer in exchange. First of all, we use the common frameworks from the methodology case. This means that we work with Scrum, of course, with the close involvement of the customer as at least one PO in this project team, but sometimes also with our own developers or domain experts. And that means that what we deliver into the project are core competencies and around the whole data engineering, data science in between, maybe the machine learning engineering everything that is necessary around it to really implement it. However, we are not domain experts, which means that sometimes we are asked about domains where we have had no previous experience and are then dependent on this domain knowledge being seamlessly represented in this team by the customer via subject matter experts. And this is not just an exchange every 23 weeks, but this person, this domain expert must be part of the team, that is a very important, yes, I wouldn't call it tooling, but that is a very important methodological requirement.
. And this innovative approach, which is also specified by Scrum with daily standups, the 2-week cycles. It embeds itself wonderfully in the Crisp DM and that's exactly what we're actually working with. That means we don't want to let these individual phases (e.g. experimentation) become too big, too weighty, too long, but rather iterate several times, i.e. we'd rather have several iterations in this cycle than stay too long in one phase, well, but now we've clarified the setup the customer says all right, I'd like to put something into production. This is the idea, work out the idea or maybe there are already concrete analyses, ideas, what can be done. But it is often the case that there is only a vague idea in the room. There is perhaps the idea to the data pot and there is perhaps an idea to the data product at the very end what should come out. But everything in between is often not yet very clearly formulated, and then it is our task to cut it up and chop it up in such a way that we get manageable morsels that we can map in this cycle and in Scrum. And that means that we first approach this explorative phase by breaking down the large task into small work packages. Their feasibility is now to be checked, i.e. the big problem is perhaps broken down into 4 or 5 unknowns and the task is now first to answer are all of these 5 work packages solvable and if so with approximately how much effort? So and then you take the first one. In the very first situations it is about, if there are already data sources where are they? Second question is then immediately can we go to the cloud? Or do we have a data protection problem here, because that is of course then lengthy discussions, but which are essential to clarify very early. So data protection, data protection, data protection. And then the actual project gets underway when it has been clarified that we are allowed to work with this data or how we are allowed to work with this data, then the actual project can get underway, this must be clarified initially and the consideration of data protection is actually almost always linked to the question: are we working in the cloud? Because as of today, we actually recommend this rigorously, so when we get into a project, we always try to position the cloud frameworks and specifically the big three, so AWS, Microsoft Azure and Google Cloud. Those are the 3 top dogs in, at least once in Europe or Germany. Well, the

representation in Germany is not quite flush with the global distribution or with the distribution in the U.S., but for us these are the 3 big clouds, and we try to initially advise the customers purely technically. What is the current status of their concerns with the knowledge they have now and without being able to go into detail about what is still to come and what questions still need to be clarified as to which of these 3 large clouds it should be for this customer. Often the customers already have something. Then the idea is also obvious to start there first. But perhaps there are already concrete requirements for data protection reasons that you have to change again or what if it is clear that we have an NLP case here and know exactly: we will probably get 80% of the solution almost for free if we use the Azure services, then of course we advise initially to complete the first POC on the Microsoft Azure Cloud and then, if necessary, to challenge performance again by evaluating other clouds and the like.
Because our approach would be, in any case, first. To use tools and frameworks and services on the cloud providers for as long as possible, until the moment when you realize that it's no longer enough to consume out of the box what's available as a service. We now have to tailor something for the customer, because when it comes to customization, the development effort is of course significantly greater. The project speed is significantly lower and for the first 80% or perhaps 70% solution, what the cloud services already provide today as a truly carefree package is often sufficient, which means that it is important for us to first convince the customer to try out the services in the clouds, to accept them and if this works in terms of data protection, and then to go into our own development, with mostly Python, to re-implement things that are perhaps still missing there. That's kind of the general way. There are exceptions, but there are fewer exceptions. A middle ground is, of course, to somehow use the cloud really only as a compute element, or to use it as storage, not to use the services, and then to go directly into custom development. But that is also becoming increasingly rare. We no longer have a pure on-promise project that does not use the cloud at all. But of course I know that it is still quite common in certain industries, because data protection and the like do not allow anything else, as things stand. Or going to a private cloud or whatever that means. Regardless of cloud or on premises or ready-made services, everything that you use is of course somehow cloud-agnostic or mobile in the sense that you don't commit to a tin, but that you work with virtualization and containers. That means Docker is used in pretty much every project.
You, I bet your question is a bit like yes open a Jupyter notebook and have a look, but no, that's not always the right answer, so yes that can be useful, but just breaking down the data locally or something, that can't work in every project. You can't answer certain questions on a section of the data or on a very small section of the data that you randomly pull somewhere, and you don't want Big Data in your Jupiter Notebook. Well, that means your RAM. That means that the preparation of the data, before you even get it into the notebook, is not trivial, so it may well be that your selection, your selection or the first feature engineering is crucial in order to be able to do exploratory analysis in the notebook.
And accordingly, it may even be totally important in the very first step to first connect this data source and what it is primarily about, if it is already clearly named, in such a way that you can do something like a selection or feature extraction and so on in a reproducible and scalable way. And then you make the first quick throw, you do that in the cloud and then the rest comes perhaps in your Jupiter notebook and yes, Jupiter Lab is of course then a top dog to just quickly explore something. But with the just said that we want to bring very quickly the things also in production or have this production thought from the beginning in the back of the head, the incentive is just not only down scripts in the Jupyter notebook, but very

quickly to think in modules and classes to generate reusable and later also in production applicable code, that is the gear is then very quickly there to switch to a smart IDE. So maybe if you are interested: as an IDE something like VS Code is naturally set or depending on what the Python requirements are exactly, there might be some preferences. Direction Pycharm for example but VSCode is already, is already a pretty good top dog and even there you can integrate Jupyter Notebooks. That's then yes a lightweight step so to speak towards production. That comes very early, that should come very early, because otherwise you have afterwards somehow 2000 lines of spaghetti code in your Jupyter notebook and that comes out something nice and then you want to build it properly, build it modularly and have it executed again so to speak and realize well, I have somehow built cabbage and turnips here and I don't even know where is up anymore, so we try to avoid that very early. And that is not always necessary. There are enough ML projects where the problem can be solved from the outset with ready-made services in the cloud. Where you don't have to do a lot of exploration, but where it's simply a matter of connecting the data so perfomantly, and then it's more a matter of data engineering or machine learning engineering performance to connect this data so that it delivers reproducible results, so that you can think about the reproducibility of the training, what data has the model seen? What release level was the model trained on, what configuration parameters were used, and so on? So that's where all these ML to production questions come in, where we also use various tools. But now I've jumped around a bit, so let's go back to the explorative phase. If there is such a phase, then of course we like to use Python and of course we also like to use a notebook first and then go into a proper IDE to build this code there in a sustainable way. And of course, all the rest around that one knows from the software engineering, is naturally added. So right from the start, a notebook should be versioned properly. Whenever we can, we work collaboratively in the projects - fortunately, we don't want to have lone warriors on projects. That means we have a kind of 4 eyes principle from the beginning, do code reviews etc. even if it's just the exploratory phase. It is still totally important that there is also a level of code quality and interchangeability. That means proper versioning right from the start and then, of course, Gitlab or Git at all is used - I can't think of any project right now where we don't use Git. Definitely the top dog. Everything that comes after that, that is, proper unit testing and an entire CI/CD pipeline, that is actually part of it from the very first moment.
As soon as code is written, even if it is only exploratory code for testing. This should be directly versioned and automatable usable from the beginning. This is important because what I just meant is that if you have built a larger Jupyter notebook and then maybe you haven't executed it cleanly from start to finish for 2 days, you can catch the wildest things and of course you don't want that, so accordingly there is always an exchange between colleagues and also automation and testing so that exactly that doesn't happen. Yes, exactly, so there too, it depends a bit on the projects, on which frameworks they then rely, but there is, I say, from this context ML Automation, that's what I would call it now, I would say mlFlow is of course very helpful, or also PyCarrot, these are the tools that are often used by us. We have also had projects where DVC somehow plays a role, but that is clearly a descending branch for us, so rather the first mentioned. Or rather the Manage Flavors on the clouds. And above that, of course, there is an entire automation in the sense of a global workflow to map this entire data flow. This happens more with AirFlow or Argo as prominent tools, also mostly with the fully managed services, on the clouds or that you deploy something yourself is actually also very, very rare. Exactly, and then it's a question of manual retraining or yes, but in the initial phase, you also want to retrain manually above all and not retrain automatically. The question is where you trigger it, whether you trigger it

from a Git pipeline, so to speak, or whether you agree with the team on a different seed, so to speak, that depends relatively individually on the projects. But there are also all forms, so that you control it via Git and the pipelines control it quite often.
Yes, okay, so here too the recommendation is clearly to maintain everything somehow in the direction of infrastructure as code, so I don't know what to use above all Terraform, in order to actually always be able to guarantee the alignment between the systems on the software side. Then you can simply work with release tags and provide all environment parameters, then you always have the right infrastructure. This means that, with the right level of expansion, production can be dimensioned larger than the development environments, or perhaps the other way around, depending on where you need to train. This can also be mapped wonderfully on the software side; we like to use the Infrastructure as Code pattern."

Q2. Do you continue experiments after deploying the ML system? If so, how do you go about doing this?
(Question was reformulated by interviewer aiming at how to prevent development code and released code from drifting too far from each other) It has to be a Docker container that everyone uses and designs equally, and then there is a very clear environment definition that somehow defines all Python and framework variables. No one should individually stray anywhere from the common release or the common development status, otherwise everything will completely fall apart.

Q3. Which tools do you use?
already answered in Q1

Q4. What are some common challenges you face in this process form a tooling perspective?
[skipped due to time constraint]

Q5. Which issues do you encounter when working with Jupyter Notebooks?
"But they are all problems that arise exactly when you stay too long in the notebook, so the solution to these problems is not easy, but it is there is a solution, namely to get away from the notebooks as quickly as possible.
Or something like a dependency management and so for me definitely does not belong somehow in the topic of notebooks, that must be regulated externally. It has to be a Docker container, which everyone uses and designs equally, and then there is a very clear environment definition that somehow defines all Python and framework variables. No one should individually stray anywhere from the common release or from the common development status, otherwise it completely disintegrates everything. The other points automated testing and so can be solved exactly by the fact that you leave Jupyter notebooks early and check in at any time again and again also in your common code versions, please not as an unreadable notebook, but so that one can also really make a diff on it. And then? And then you've ensured collaboration and ensured continuous integration so don't linger too long in the Jupyter notebook solves yes, I would say now all of these points relatively reliably. It's still an overhead. Of course, it's easier to just let everyone tinker for themselves, but that's a technical debt that pays off from day 2. So I wouldn't recommend doing that at all. Exactly and experiment, tracking, etc. in no case build yourself, so these are exactly the pitfalls that you then get later at the latest a quarter later around the ears. So use frameworks like mlFlow and so on. And please don't do it from the Jupyter notebook, but already in an at least partially automated environment. Oh well, the Jupyter Notebook is

really such an intraday element for me, so when I'm sitting on the train, I start a dump from the database or something and can just quickly try things out on the data frame or something. But that must not stay there. Theoretically, I think it's a good idea to throw away notebooks almost every day. And save what you've learned in meaningful code and check it in and put it back into the whole release pipeline, otherwise you run into the problems.
Yes yes, the whole thing is an honorable goal of course. But yeah, it's really super exhausting in the beginning. And it doesn't always succeed and you're often away from my intraday so that you do it daily. But then let it be 2 or 3 days, then that's still good and you can, you just have to want to do that.
Oh, if you stay in Jupyter Notebooks forever then you can slap on another half year of project time on top of that to fix the whole thing. You are actually already finished and only need to bring it into production and that takes another half a year, not a good idea.
"