# From Prompt to Model: Fine-tuning when you've already deployed LLMs in prod w/Kyle Corbitt

32 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=4EPZZKVRXC4
https://www.youtube.com/watch?v=4EPZZkVrXC4

SUMMARY

*The discussion focuses on best practices for deploying fine-tuned models in production, emphasizing that fine-tuning should not be the default approach unless specific conditions warrant it. The speaker advocates starting with prompting for faster iteration and better results, only transitioning to fine-tuning when quality, latency, or cost issues arise.*

- *The first commandment is to avoid fine-tuning unless necessary; prompting should be the default.*
- *Prompting allows for rapid iteration and adjustments, leading to better outcomes.*
- *Fine-tuning is justified when prompting fails to meet quality standards, requires faster responses, or becomes cost-prohibitive at scale.*
- *Establish a baseline with prompting before fine-tuning to evaluate improvements effectively.*
- *Analyzing user data helps identify input distribution and informs model adjustments.*
- *Avoid excluding problematic data from training; instead, address issues directly to improve model performance.*
- *Maintain a representative test set to ensure evaluations reflect real-world performance.*
- *Continuous evaluation post-deployment is crucial to detect data drift and maintain model accuracy.*

the topic is you know deploying fine T models and production the the first commandment and the most important commandment of deploying fine tune models in production is you should not do it okay all right this is obviously not universally true or else I wouldn't be giving this talk but I think it's a good default so specifically if you have an existing flow that is working for you and you figured out you know a pipeline with with a prompted

model that or or if you like are able to figure out a pipeline with a prompted model that probably should be the default there's a lot of advantages there a big one is your iteration speed is much faster if you notice like you know something's wrong you can play with it you can update the prompt really quickly and it just leads to a better experience versus fine-tuning and which you know we I I think there's a

lot of tools and open pipe is one of them but you know we we try and get the experience as good as possible but really honestly you can't really beat the experience of like you just change the words and rerun it and you get different or better results so the default should be don't bother with fine tuning unless one of like some very specific things are true okay okay and this this is what I just covered start with prompting you

know if you want to get a little bit more you know fancy than just like Rog typing out the instructions very often doing something like hey coming up with three or four examples that are really high quality and throwing them in your prompt or even a little bit faster than that maybe you have a bunch of examples that you know are good and you can sort of like use a rag thing where you you grab the examples that are

most similar to the you know to the exact document you're or whatever you're analyzing right now these are all strategies I would reach for before I reach for fine tuning okay now all of that said there are some very specific and very good reasons why you would go past that point and why you you should actually do fine tuning and and in my experience there's there's three dominant ones so the first reason is is quality right so you often find

that there's only so far you can get with just prompting as far as guiding a model towards a specific outcome and if how far you can get with that is not all the way to to where you need to be to provide a good experience that's a great reason to do fine tuning another really good reason is because when you're doing fine tuning typically the I guess you could say like the prto frontier of like how far you

can push a given model or a given size model like performance-wise is much further out with fine tuning than with prompting and so what that means is you can move to a much smaller model for a given Quality Bar and the upshot of moving to a smaller model is that you can get responses much faster so there are a number of use cases we have users like this where you know you're doing real time you know

you're like doing phone system work or even chat work where you just want to get a really fast response or kind of like the double whammy of you you want to get a user's response back fast but you've got like this agent agentic flow where it's you know you've got several different prompts you're running in sequence to kind of like figure out like all the stuff that needs to happen and then to get the response back to the

user and of course every one of those takes a latency hit and so if you've got several of those running and and the user is waiting you know you're going to have a much better experience if those are going faster and fine tuning can can get you to that spot and then the final one and this is actually the one we see pushing people the most often is about cost so in a lot of cases there there's a lot of use cases where

gp4 can do a fantastic job and you know latency is not an issue you're just using it for classification or whatever but like once you're doing this on hundreds of thousands or or potentially millions of calls every day it just gets way too expensive from a pure unit economics point of view and so this this is a very strong reason why people will go to find tuning because again you can move to those smaller model sizes

and still get a really strong a really good performance and and of course the cost per token is is much lower at those lower model smaller model sizes Okay so we've established that these are that one of these things is true and again you should not find tune unless one of these things or some other reason that's a very good one is true but let's say we've established this right that the with promp either we can't hit quality or latency

or cost okay so now let's go into the actual tuning part and and talk about what we're doing there actually trick question or or trick statement we're still not going to go to find tuning okay so even if you know that this is true even if you know it's like hey I am not going to be able to deploy in production because I know I just know that it's going to be too expensive when I do this at scale or I just know that

like I can't hit my Quality Bar you should still start by writing a prompt that gets you as close as you can and I'm going to explain why that's important or I mean you know you can get away without doing this but why I think this is this is for most people for most flows the way you should do it so the first thing is it gives you a baseline right it's like okay if if I

want to know if my fine tune model is worth it if I should be investing time in this like I need to know in the alternative where it's just prompted like what I'm coming from and what I'm trying to improve on and so that's really important and as we're getting later on where we're talking about evals we're talking about like actually the user experience you're providing this gives you something to compare it to and just make sure hey is

is is all the time and effort I'm investing in doing this like actually giving me a return this next point is I think a little bit of a subtle one that people don't think enough about but I think a really critical one and that is trying to solve your problem with a prompted model can give you very good information about whether the task is possible so this is this this is actually something we see very frequently where someone will come to

us and and you know they're trying to solve a problem with fine tuning and they haven't either they haven't tried or they haven't been able to get it working with prompting yet and we'll have a conversation with them and say like okay you know explain to me like can can you do this with with prompting they'll say oh no it's because you know the model like we have this like very you know weird schema that we're trying

to get the model to output and it's just like prompting can't get it there or some other reason why they think it doesn't do it but then they'll and and so I'll say okay well we need we need data to find tune and often they'll say oh no that's that's no problem we have a bunch of data like we have data we can train it on this is not a problem what I have found in practice is that it is

very often unfortunately I'd say like more often than not the case where even if you think you have labeled data that should work for this the data you have if if it's not good enough to like you know stick in drag pipeline or something like that and make it work with prompting there's a very high chance there's actually not enough signal in that data to do what you want it to do and there was actually a

fantastic example that I think Dan gave in in the first lecture here about a logistics company where you know that you you you may remember if you watched that lecture he was talking about they were trying to go from the description of an item to the estimated value of the item and I think there was a good assumption there like a hypothesis that like Well the description of the item should tell you enough about the item that like a model

that understands a lot about the world should be able to guess how much it's worth but in practice what they found is that it actually didn't it didn't capture enough information there was enough Randomness or or reasons why people weren't putting the right thing in the value box anyway there actually was not enough information in that training data whereas if you're going with a prompted flow and you're trying to get that working first you can you can find out very quickly like okay

I just don't have the data here you know the model's not able to figure this out this point is not a universal like it definitely is possible to train a model that through fine-tuning that can that can like have great performance on like your proprietary data set or whatever that there's like no way that a prompted model could do so definitely not a hard and fast rule but I found like you know that's if if if you can go

in this direction it's going to make your life much better so just kind of like a heris on that point is the so so if you do find that you're able to successfully write a prompt that is able to at least you know more often than not do a good job on your task then there's like a 90 plus percent chance that you will be able through fine tuning to improve that further on the metrics you care about so whether that's

latency quality or cost there's like a very good chance you can get this working whereas on the other hand if like there is no way to formulate your task in such a way that a prompted model could ever work and you're kind of just hoping that it can learn from the data I mean that still can work but you're definitely playing in hard mode at this point like this is now a hardc data science project like you have to

figure out is you know is there actually enough signal in this to solve the problem and and it it just gets much more complicated and in a lot of cases it turns out that there actually isn't that your data is not is not clean enough or well labed enough or whatever to to learn what you wanted to and so there's a high chance of failure so so you want to be you want to be in this

regime you you really want to be in the regime where you know it works with prompting and then there's a very high chance that fine is going to be able to to juice your your returns and get you way better in a way better place you want to avoid being in this other regime Okay so we've got a prompt and and now we're going to find tune and okay I guess this this slide is something I just wanted to share quickly

the the conceptual way I think about it the way I encourage folks in your situation who are thinking about deploy and production to think about it is like you know you're you're going with the Prototype you're basically when you're when you're iterating fast when you're trying to figure out is this even something like possible to do is this something that's providing value is this something that you know with our whole whatever whatever app or flow or

or whatever it is you're building and you're trying to see if it's going to work or not and if it's going to provide value just do all that part with gbd4 don't worry about fine tuning and then once you've got something that actually works that scales that you have a sense what people are using it for that's the point where you're going to drop in fine tuning that's that's just kind of like the general mental model that

again not Universal there's there's reasons and and times when it makes sense to go straight to find tuning to model if you can't get prompton working but in general this this is the flow I would recommend trying to make work by default okay so we have so let's say we now have a prompt deployed in production we have you know people using it people prototyping playing with it what's the next step well you got to look at

the actual data and this is going to teach you so much both on on like how good a job your existing prompted model is doing super important but also like on the types of things that people are using your product or your model for and that's actually the part that I find personally is the most useful part of reviewing data is it just gives me a very a much better sense of like okay what is it just GES me a feel for like

what my in input distribution looks like and without that information like you can make assumptions about you know the types of tests you should write the the the types of like even the types of models you think will work well or poorly based on like you know just like how hard you think the task is the types of things you think this is being used for that can be very wrong so you just got to like look through it this is

there's there's there's no magic here the right way to look through it varies a lot like very often in whatever system you are writing you have some sort of natural UI whether it's a chat conversation interface whether it's like some kind of classification system and you're you know putting documents in different things so so if you do have kind of like an existing UI then that's probably the right place to look at it in if not if for whatever

reason that there's not a good way then there are tools out there you can use and and open pipe is one of them but but that'll just give you a nice format at okay this is what the input looks like this is what the output looks like let's let's click through let's go through 20 50 of these and just get a sense of of what our data looks like and you're wanting to look at both the

input and the output you want to get of the outputs any good as well of course but honestly like I said I find a lot of the value here is is just getting a really good sense of of my input distribution Okay so we've looked at our data we have a sense of what's going on what's next so okay so this next one is like I I need to explain what I mean by this so you actually once you've

once you've looked at this data once you understand okay this data is good now you have to like act that that is the data you should use and let me like the failure case I see here is there's a lot of there's there's like a tendency where sometimes people will look to their data and they'll be like oh I noticed this like particular class of data the that the model I'm using in production right now it does a

bad job on and so what they'll do is they'll go and they'll like drop that out of their data set before they find tune on it because they're like well I only want to find tune and and they you know sometimes people have like a fancier automated way of doing this where they say okay well you know I I'm going to collect user thumbs ups and thumbs down on my prompt and then the ones that get thumbs up I'm going to

go ahead and F tune on those and the ones that get thumbs down I won't and I'm not going to say that never works like that definitely can work the failure mode that I see here that does happen though is if you're rejecting all the places where the model's doing a bad job there's a real danger that like there is some correlation between the things that does bad on it like a specific like space of your inputs or or

something that like you actually do care about and if you don't find tune on that then you're just going to like you're just going to do a bad job there so so the real solution here is figure out why the model is doing a bad job figuring out what where in your input space is it doing a bad job and then do one of like several things you can manually relabel it you can you can like you know

just like spin up a a relabeling UI and and there's several of them out there that are pretty good and kind of like type out what should be you can like try and fix your instructions often times I find that's the best fix is like as you're manually reviewing the data you're like oh it's doing a bad job here and then you play around with it and you fix your instructions and and you can get it to a place where it's doing a

much better job there so that's very often the right way to do it but the main thing is you don't want to just like drop those ones and just F tune on the ones that it did a good job on because like there's a very high chance that means you're just like chopping off big chunks of kind of like your input space and then when you have your fine two model that you're deploying production it's never seen this data

before and it's also just going to do a bad job in that area so so that's anyway no no no shortcuts there now I'm going to contradict myself a tiny bit which is if what you find is the model is mostly doing a good job and like 10% of the time but it's kind of like a random 10% of the time it like forgets an instruction or get something wrong or something like that what have I I

have actually but it's not like super correlated in like this one area it always gets it wrong it's more just like well sometimes dd4 it's non-deterministic it messes stuff up once in a while I have actually found that when that is the case it actually doesn't really hurt to train on that data and I this is what I'm saying right now is like very controversial and I'm sure that like you know like I could have a great debate

probably with with people on this call right now about like whether this is you know like how how big of an issue this is but I'm just saying from my own experience what I found is like you can very common like basically these models are quite smart and they're quite good at generalization as we're getting into these larger models and and I'm talking really the advice I'm giving right now really is for like you know I'm talking about like LMS I'm talking

like for8 billion plus parameter models I'm not talking about like the really small ones but for these big ones I actually find that like the training process itself to some extent like does a form of like regularization or normalization where even if the input data is not perfect as long as it is like pretty good on average you can actually very commonly see the model outperform even the the model that was used to to train it originally

because of that sort of regularization effect and so we see that very commonly on our platform we have lots of users who are training their fine tune models directly on gbd4 outputs and then they'll actually run evaluations between gbd4 and their fine tune model and consistently see for for a lot of tasks that their fine tune model is doing better than gbd4 and I again I think this comes down to that regularization effect where gbd4 will

just like you know 10% of the time make a dumb error and and part of the training process it sees those but it sees the 90% good ones and it learns you know that that basically the the goodness so anyway this all this to say don't stress too much if there's like a few bad examples I think in many cases you can get great results anyway and it may not be worth the effort of like manually reviewing all

whatever 5,000 or 10,000 examples you have okay so let's say we've got you know we're pretty happy we we've rued our data like okay you know at least like 90% of these look look good this is basically doing what I want my model to do okay so next before we actually do our training very important you have to pull out a test set okay and this is this is not news to anyone who you know has has a background of machine

learning but I do actually see this as something that that that sometimes people don't do or or often I'll see people who have a test set which is kind of like weirdly constructed so they'll like pull out like I don't know they'll come up with like 10 random examples because they saw these are the ones the prompted poorly on or these are ones that like for whatever reason you know a customer complained about this so so

they have like a way of constructing a test set that is not representative of the input data I guess is the way I would put this and I think that's totally fine like I think having a test set like that of like specific cases that you know are weird Corner cases and you want to make sure it's doing well on like that's that's a that's actually great there's nothing wrong with that the problem is if you are testing on

that exclusively and you are not also having a test set which is just like a basically like a random subsample of your inputs then you can be like you you you can you can have you can think you're doing well and really not be doing well like you really want to have a test set which is just like hey grab 5% grab 10% of my data at random don't train on it and and and use that as the input so highly recommend

doing that this is yeah just kind of standard stuff from machine learning but but again something I see people that don't have a lot of experience fine tuning maybe not realizing is is is really important okay so now let's talk about the actual model that you should be fine tuning so one nice thing is if you've got and I know that you've had you know like you youve you've already had a chat with wing from Axel

a really nice thing about Axel or you know like hugging faces like sft trainer things like that is and and just like the hugging phas Transformers ecosystem in general is like there are a lot of it's pretty easy if you get like a fine tuning pipeline set up for one model to like throw in several different ones and kind of like run them side by side and as long as your data set isn't huge like the cost is not prohibitive like we're

talking maybe a few dollars in many cases per fine tuning run and so that that's great because it makes it a lot easier to kind of like try different things but kind of as like a a rule of thumb or a place to start so this this is a chart I put together actually for a different presentation but I think it's kind of representative of the ecosystem and and some of the models that people are find tuning commonly

and so the sense here so so this is this is actually based on real data the the what what this is normalized to is the number number of training examples it took to match a certain performance threshold so basically for the specific test that this is I I looked at like okay how good do GB before do on this and then for each of these other models like how many training examples did I have to add

until it was able to match gbd 4's performance on this specific task and the answer to that question is Task dependent but but this is kind of like this this gives you an overview and what I find is you know in general if you've got like a few dozen examples like you can very often get like llama 370 billion to match gbd4 on your example and then as you start getting to smaller models it does does take more

training data it takes a wider input of training data and again this is very task dependent there are some tasks where like it doesn't matter how much training you do you're never going to reach gb4 performance I actually find that that's like definitely the exception for most things I see people running in production I find that Like These Fine two models actually it usually can pretty easily MCH gbd Force performance if you have a good a

good training set the place where I actually see most people like coming in when they're actually doing this in production really The Sweet Spot is like this 8 billion 7 billion 8 billion parameter model and so like things like llama 38 billion M 7 billion I see that as like a really nice sweet spot because the amount of training data you need to get good results out of that is not overwhelming you know if you have like a

thousand examples or so you can typically get really good performance out of this and and at the same and and so if you're running in production like we were talking about earlier anyway like hopefully you do have like a few thousand examples that you've kind of gotten that have gone through gp4 anyway so hopefully you're in a good spot and and the cost savings are really significant so actually this I think yeah I didn't update this with the

latest inference costs you can get online but this should actually be lower like you know the the cost you're seeing per token for like a l 3 a billion is somewhere in the 15 or 20 cents per million tokens versus gbd4 even GPD 40 is I think about 50 times that so it's a huge shavings you're getting there and and you can get a really good performance in that you know, to 5,000 example range so that's where I

often recommend people go but again these training runs the nice thing is they're they're really cheap to do especially if you're in that like you know Le 5,000 or less example thing and so so why you know you can try a 70 billion model and see if it does better you you can even go smaller you can try like a 53 and and see how it does and like ultimately like it's pretty easy test to run but but this is

probably what where where might have would be for for a lot of tasks at least okay so now we're going to talk about evaluations evaluations obviously are a huge subject all on their own and I'm not going to go super deep into this I think probably there's other segments of the course where you're going to be talking about this more but I do think there's some there's like an interesting framing here which is which is not one that I hear people talk about

that much and I think is pretty important because in my mind there's actually two different kinds of evaluations both of which are really important and so the first one are fast evaluations and fast evaluations are ones that you can run like in your training Loop or like even if you're doing like prompt engineering like as you're updating the prompt these are evaluations you can just like update the prompt run and then immediately see did it good did I do a good job or not so

the these should be relatively fast to run these should be relatively cheap to run and not require a lot of outside input to get good results the where I personally have found a really good sweet spot for this this category these fast evaluations is using llms judge and so that's kind of like the default that's where I would start is you know basically like asking gbd4 or asking like a jury of like you know gbd4 and Cloud 3 and maybe another model or

something to say okay you know this is the task is is the input I had and then these are you know these are two different outputs from two different models and and there's like some tricks here like you have to randomize the order because there's there's like a slight preference for the former one and and you know if you're using the same mod as judge and and as one of the entries like it has a preference for

itself so so there's there's like a little bit of subtlety here I think there's good libraries out there that that like can help you do this that are built in you know we also like on open pipe this is sort of like the default evaluation we give people but the main point is these things are like cheap enough to run if you're running it against say you know 50 or 100 examples that you can just like you

can update your prompt and rerun it or you can find a new model and rerun it and really quickly get a sense okay is this is this like plausibly doing helping me or not and I think that having something really fast like that that you can run quickly and get a sense okay am I am I on the right track or not is super critical because otherwise you know if you get to and I'm going to

talk about the other kind of evaluation just a moment but like basically if you've got like these these slower evaluations that require you to run in production your feedback cycle is so slow that it's just a lot harder to to make progress and to get where you need to go so I really recommend investing in fast evaluations you know I think El Jud as judge is right is great if you're doing there's also like for specific tasks other evaluations

that can make a lot of sense like if you're doing like a classification task or something then you can just get a golden data set and calculate an F1 score or something like that so anyway but but the the high level point is find something that you can run fast and and have a quick inter Loop and and can make help you figure out you're in the right direction okay now the other kind of evaluation which is also super important

are like outer loop evaluations slow evaluations production evaluations and you can call it different things but the idea is that this is the one that is actually measuring the final result that you care about right and so this is something like if you're writing a like a chat bot like a customer support chat B it's like okay what percentage of customers came out of this feeling like their problem was resolved right and so these evaluations I don't think

there's like I mean there definitely is not a one-size p all it's it it really basically comes back to like what is the outcome you know the business outcome or or the product outcome that you're trying to drive and figuring out how can I measure that and so you really these are really critical as well because you know the fast evaluations even if a call looks better in isolation maybe it's if like a specific model output looks better in

isolation maybe it is not maybe there's there's like you know like some other interaction with other piece of the system that's like giving you a bad result or maybe like the elus judge is like not perfectly accurate and it's not actually measuring or maybe it's like once you deploy it to production like you're quantizing it or something and and there's like some disconnect in like the actual way it's running so there's all these reasons why the sort

of fast Dev vales you were doing before might not tell you might not give you the full picture and might not be perfectly accurate and so you really want to have that Outer Loop and make sure that you were going in the right direction okay so I have a couple of examples here just from open AI with chbt you know in their particular case I think they measure how often do people regenerate how often do people give like

a thumbs down they also have like a more concrete flow which which they rarely put up but I have seen it a few times where you know you ask a question it'll just give you two responses side by side it'll let you choose which one is better and I think again obviously it's really dependent if you're not writing a chat bot then probably this is not the right form but I do think it's that you think

about for your specific problem how are you actually going to measure that you're doing a good job and that it's it's driving the outcome that you actually care about okay so we're almost we're almost getting through this we're to the ninth commandment out of out of 10 this one is really important so hopefully like in this previous one you've written these these slow evaluations you're actually looking you have some way of measuring at least somewhat objectively or repeatably

like how well you're doing well once you've deployed your fine T model you really have to keep running those on a constant basis because if you don't what you're going to see is something is going to change about the world there's going to be some difference in the types of users you're getting or the things they're sending you or there there's just like so much that can change out there that if if you're not continually measuring how well this

is doing like you're going to have an and like sort of the the term machine learning practitioners use for this is data drift which can come from a lot of sources but the main point is like over time it's very likely that your model is not going to be as well adapted to the input as as it should be and like one so one one interesting concrete example in in our case we had a customer who was doing

basically structured data extraction from from from these call logs actually transcripts and they noticed that they had this this eval that they were running in sort of this this slow Loop and they noticed that their that their responses were getting worse right around January of this year so I guess five months ago and so it wasn't huge it was like a you know I don't remember exactly what the numbers were but it went up from like a you know

1% error rate to like a 5% error rate where where was like hitting their things and and wasn't working and so so they ended up looking into it and and they shared with us what they found which which I thought was fantastic which was the the data extraction there were there were like a number of fields they were trying to extract from these call logs and one of them was like a specific date so it was it was

people calling in about it was call logs about people that like mortgages basically and and trying to get the due date on their payment whatever so one of the things they were extracting was like what is the specific date that like this the next payment was due and what they found was because their model had been fine-tuned entirely on data from 2023 not in all cases but in like 5% of cases even though it were now in 2024 it

would just kind of like you know it would get the date like the month and day right but then it would like put the year as 2023 in in the extracted date instead of the year of 2024 because it was just like in every single case of the training date it was the the year was always 2023 didn't see any other examples and I didn't do that every time like it was smart enough in most cases to figure out okay this you know

we should be pulling out the 2024 but anyway that that was starting to happen so all they had to do was retrain a model with a few extra you know I don't remember how to put in they put in like you know 10 or 12 2024 examples and that was that was plenty to clear it up so anyway just an example you never know where this stuff is coming from but you do have to keep measuring

to see if things get worse and okay and so finally the last one is I I frame these as Commandments I tried to give disclaimers as I was going through but I think it's really important to realize that the most important thing is that you understand your problem that you understand your data you understand what you're trying to solve and then you figure out if like what the right way is to solve it so I think the flow I

describ is really really helpful I think there are other ways to do it that can also be effective but but but but I do think that like this is a good place to start especially if you haven't done this before and you want to just maximize the chances that you're able to do it successfully so anyway yeah it's like it's like it's like Pirates of the Caribbean right where the the pirate says like you know well the

pirate code's really a code right it's it's a guideline anyway same feeling