03:23:10 validation data split, validation loss, sampling revive

adjusted the code so that we're evaluating on the validation split so creating the Val loader just by passing in Split equals Val that will basically create a data loader just for the uh validation Shard um the other thing I did is in the data loader I introduced a new function reset which is called at init and it basically resets the data loader and that is very useful because when we come to the main training Loop now so this is the code that I've added and basically every 100th iteration including the zeroth iteration we put the model into evaluation mode we reset the Val loader and then um no gradients involved we're going to basically accumulate the gradients over say 20 steps and then average it all up and print out the validation loss and so that basically is the exact same logic as the training Loop roughly but there's no loss that backward it's only inference we're just measuring the loss we're adding it up everything else otherwise applies and is exactly as we've seen it before and so this will print the validation laws um every 100th iteration including on the very first iteration uh so that's nice that will tell us some amount some a little bit about how much we're overfitting that said like uh we have roughly Infinity data so we're mostly expecting our train and Val loss to be about the same but the other reason I'm kind of interested in this is because we can take the GPT 2124m as openi released it we can initialize from it and we can basically see what kind of loss it achieves on the validation loss as well and that gives us kind of an indication as to uh how much that model would generalize to 124 M but it's not an sorry to fine web edu validation split that said it's not a super fair comparison to gpt2 because it was trained on a very different data distribution but it's still kind of like an interesting data point and in any case you would always want to have a validation split in a training run like this so that you can make sure that you are not um overfitting and this is especially a concern if we were to make more Epoch in our training data um so for example right now we're just doing a single Epoch but if we get to a point where we want to train on 10 epochs or something like that we would be really careful with maybe we are memorizing that data too much if we have a big enough model and our validation split would be one way to tell whether that is happening okay and in addition to that if you remember at bottom of our script we had all of this orphaned code for sampling from way back when so I deleted that code and I moved it up um to here so once in a while we simply value validation once in a while we sample we generate samples and then uh we do that only every 100 steps and we train on every single step so that's how I have a structure right now and I've been running this for 10,000 iterations so here are some samples on neration 1,000 um hello I'm a language model and I'm not able to get more creative I'm a language model and languages file you're learning about here is or is the beginning of a computer okay so this is all like pretty uh this is still a garble uh but we're only at ration 1,000 and we've only just barely reached maximum learning rate uh so this is still learning uh we're about to get some more samples coming up in 1,00 okay um okay this is you know the model is still is still a young baby okay so uh basically all of this sampling code that I've put here everything should be familiar with to you and came from before the only thing that I did is I created a generator object in pytorch so that I have a direct control over the sampling of the random numbers don't because I don't want to impact the RNG state of the random number generator that is the global one used for training I want this to be completely outside of the training Loop and so I'm using a special sampling RNG and then I make sure to seed it that every single rank has a different seed and then I pass in here where we sort of consumer in the numbers in multinomial where the sampling happens I make sure to pass in the generator object there otherwise this is identical uh now the other thing is um you'll notice that we're running a bit slower that's because I actually had to disable torch. compile to get this to sample and um so we're running a bit slower so for some reason it works with no torch compile but when I torch compile my model I get a really scary error from pytorch and I have no idea how to resolve it right now so probably by the time you see this code released or something like that maybe it's fixed but for now I'm just going to do end false um and

I'm going to bring back toor compile and you're not going to get samples and I I think I'll fix this later uh by the way um I will be releasing all this code and actually I've been very careful about making get commits every time we add something and so I'm going to release the entire repo that starts completely from scratch all the way to uh now and after this as well and so everything should be exactly documented in the git commit history um um and so I think that will be nice so hopefully by the time you go to GitHub uh this is removed and it's working and I will have fixed the bug okay so I have