01:02:00 data loader lite

and I wrote a little data loader light um so what this data loader does is we're importing the token up here we're reading the entire text file from this single input.txt tokenizing it and then we're just printing the number of tokens in total and the number of batches in a single Epoch of iterating over this data set so how many unique batches do we output before we loop back around the beginning of the document and start reading it again so we start off at position zero and then we simply walk the document in batches of B * T so we take chunks of B * T and then always Advance by B * T and um it's important to note that we're always advancing our position by exactly B * T but when we're fetching the tokens we're actually fetching from current position to B * t + 1 and we need that plus one because remember uh we need the target token um

for the last token in the current batch and so that way we can do um the XY exactly as we did it before and if we are to um run out of data we'll just loop back around to zero so this is one way to write a very very simple data loader um that simply just goes through the file in chunks and is good enough for us uh for current purposes and we're going to complexify it later and now we'd like to come back around here and we'd like to actually use our data loader so the import Tik token has moved up and actually all of this is now useless so instead we just want a train loader for the training data and we want to use the same hyper parameters for four so B size was four and time was 32 and then here we need to get the XY for the current batch so let's see if copal gets it because this is simple enough uh so we call the next batch and then we um make sure that we have to move our tensors from CPU to the device so here

when I converted the tokens notice that I didn't actually move these tokens to the GPU I left them on CPU which is the default um and that's just because I'm trying not to waste too much memory on the GPU in this case this is a tiny data set and it would fit uh but it's fine to just uh ship it to GPU right now for for our purposes right now so we get the next batch we keep the data loader simple CPU class and then here we actually ship it to the GPU and do all the computation and uh let's see if this runs so python train gbt2 pi and what do we expect to see before this actually happens what we expect to see is now we're actually getting the next batch so we expect to not overfit a single batch and so I expect our loss to come down but not too much and that's because I still expect it to come down because in the 50257 tokens many of those tokens never occur in our data set so there are some very easy gains to be made here

in the optimization by for example taking the biases of all the loits that never occur and driving them to negative infinity and that would basically just it's just that all of these crazy unic codes or different languages those tokens never occur so their probability should be very low and so the gains that we should be seeing are along the lines of basically deleting the usage of tokens that never occur that's probably most of the loss gain that we're going to see at this scale right now uh but we shouldn't come to a zero uh because um we are only doing 50 iterations and I don't think that's enough to do an eoch right now so let's see what we got we um we have 338,000 tokens which makes sense with our 3:1 compression ratio because there are 1 million uh characters so one Epoch with the current setting of B and T will take 2, 600 batches and we're only doing 50 batches of optimization in here so we start off in a familiar territory as

expected and then we seem to come down to about 6.6 so basically things seem to be working okay right now with respect to our expectations so that's good okay next I want to actually