03:56:21 shoutout to llm.c, equivalent but faster code in raw C/CUDA

that I wanted to briefly show you is that of course what we've built up today was building towards nanog GPT which is this repository from earlier uh but also there's actually another nanog GPT implementation and it's hiding in a more recent project that I've been working on called llm Doc and lm. C is a pure Cuda implementation of gpt2 or gpt3 training and it just directly uses uh Cuda and is written as Cuda now the nanog gbt here acts as reference code in pytorch to the C implementation so we're trying to exactly match up the two but we're hoping that the C Cuda is faster and of course currently that seems to be the case um because it is a direct optimized implementation so train gpt2 Pi in LL M.C is basically the nanog GPT and when you scroll through this file you'll find a lot of

things that very much look like um things that we've built up in this lecture and then when you look at train gpt2 docu uh this is the C Cuda implementation so there's a lot of MPI nickel GPU Cuda cc++ and you have to be familiar with that but uh um when this is built up we can actually run the two side by side and they're going to produce the exact same results but lm. C actually runs faster so let's see that so on the left I have pytorch a nanog GPT looking thing on the right I have the llmc call and here I'm going to launch the two both of these are going to be running on a single GPU and here I'm putting the lm. C on GPU 1 and this one will grab uh gpu0 by default and then we can see here that lm. c compiled and then allocate space and it's stepping so basically uh meanwhile P torch is still compiling because torch compile is a bit slower here than the lm. C nbcc Cuda compile and so this program has already started running

and uh we're still waiting here for torch compile now of course uh this is a very specific implementation to gpt2 and 3 a pytorch is a very general neural network framework so they're not exactly comparable but if you're only interested in training gpt2 and 3 lm. C is very fast it takes less space it's faster to start and it's faster per step and so P started to Stepping here and as you can see we're running at about 223,000 tokens per second here and about 185,000 tokens per second here um so quite a bit slower but I don't have full confidence that I exactly squeezed out all the juice from the pytorch implementation but the important thing here is notice that if I Aline up the steps you will see that the losses and Norms that are printed between these two are identical so on the left we have the pie torch and on the right this C implementation and they're the same except this one runs faster uh so that's kind of I

wanted to show you also briefly lm. C and this is a parallel implementation and it's also something that you may want to uh play with or look at and um it's kind of interesting