Cublas should be the most optimal approach. Cublas>gpu tiled> gpu naïve>cpu naïve. When running small It’s not surprising that cpu matrix multiply takes longer time as matrix size increases. But the fact cublas is slower than gpu tiled and gpu naïve shocks me. By the way, gpu tiled has approximately the same running time as gpu naïve. My guess is that the matrix size is small, which is up to 200. Cublas is able to handle greater sizes but other implementations might get stuck.

Cpu naïve, gpu naïve and gpu tiled went smoothly, but I got stuck on cublas since I don’t understand the code because there’re lots of error checking and other function calls, especially we’re given three different pieces of information—slides, nvidia, github—I don’t know which one I should refer to. After discussing with Elaine, I finally had a clear mind of what to do next. I found the important part from the whole code chunk, and then revised that based on my own initialization of matrix. I started to know how to implement this function. When I thought everything was perfectly settled, the matrix prints all 0. I started to debug that, which took me a longer time. Even if I checked each line, I was still not sure where’s the bug. Then I asked chatgpt to help me find the bug, which turns out, that the type of device matrixes should be float\*\* instead of float\* because they are the pointers of float\*. Next time I will pay more attention to the memory allocation and pointers. I think that I should also include some error handling just like what the git hub code has, but I’m still not familiar with cuda errors.