📘 Lecture Notes: Tensors, GPUs, Kernels, Quantization & GGUF

1) What Are Tensors?

A **tensor** is an N-dimensional data structure (0D → scalar, 1D → vector, 2D → matrix, 3D+ → higher dimensions).
Tensors hold **homogeneous datatypes** such as `float32`, `float16`, `bfloat16`, `int8`, etc.
PyTorch does **not** expose 4-bit or 2-bit tensor types, though such formats exist internally in quantized backends.

---

2) Why Tensors Are Fast on GPUs

GPUs are made of **thousands of ALUs**, allowing massive parallel computation.
CPUs also do parallelism, but Python encounters the **GIL**, which prevents true multithreaded parallel compute.
PyTorch operations bypass the GIL (because they run in C++/CUDA), enabling fast GPU execution.

---

3) GPU ↔ CPU Transfer Bottleneck

GPU VRAM has extremely high bandwidth.
Transferring data to CPU RAM via PCIe is **much slower**, so keep tensors on the GPU once moved.
Minimizing transfers is critical for performance.

---

4) The Word "Kernel" Has Many Meanings

The term **kernel** is overloaded:

- OS Kernel → interface between hardware & software
- CNN Kernel → a filter used in convolution
- CUDA Kernel → a function executed in parallel on the GPU
- GEMM / Matrix-Multiply Kernels → highly optimized routines in NVIDIA libraries

CUDA kernels are written in C++/CUDA, so **GIL has no effect** on GPU execution.

---

5) SIMD vs SIMT

- **SIMD (CPU)**: Same Instruction, Multiple Data (vectorized operations on one core)
- **SIMT (GPU)**: Same Instruction, Multiple Threads (warp of threads executing in lockstep)

GPUs implement massively parallel SIMT execution.

---

6) Matrix Multiplication Complexity

Theoretical complexity:

- Classical algorithm: **$O(n^3)$**
- Fast algorithms (Strassen, Coppersmith-Winograd): ~**$O(n^{2.3})$**

Parallelism reduces wall-clock **time**, but not theoretical complexity — it only *feels* like O(1) because multiple operations occur simultaneously.

---

7) Memory Layout of Common Datatypes

Type	Bytes
FP32	4
FP16	2
BF16	2
FP8	1
INT8	1

Smaller dtypes reduce memory footprint and improve throughput.

---

8) Additional GPU Optimizations

Deep learning frameworks use advanced techniques such as:

- **FFT-based convolution** → efficient for large kernels
- **Winograd convolution** → fewer multiplications for 3×3 kernels

These reduce arithmetic cost and improve GPU throughput.

---

9) Quantization Techniques

Quantization reduces model size and speeds up inference.
Common methods:

- **PTQ (Post-Training Quantization)**
- **QAT (Quantization Aware Training / Fake Quantization)**
- **Weight-Only Quantization (W8A16, INT4 weights, etc.)**

Quantization is handled by backend libraries like TensorRT, FBGEMM, and XNNPACK.

---

10) GGUF Format (LLM Quantization Storage)

GGUF is *not a quantization method* — it is a **model format** used by `llama.cpp`.
It stores:

- Model weights
- Tokenizer
- Metadata
- **Quantized weights**: Q2, Q3, Q4, Q5, Q6, IQ2, IQ4, etc.

GGUF enables running very large models efficiently on **CPUs, WebGPU, laptops, and small GPUs**.
Think of it as a **container that holds quantized models**, not the technique that performs quantization.
