EECS E6893 Big Data Analytics

HW4: GPU Programming, Deploying LLMs

Tutorial · Running & Serving LLMs · Mini-Project

EECS E6893 · Big Data Analytics · Fall 2026 · TA: Yanhao Bai yb2630@columbia.edu

Agenda

GPU programming: SIMT, CUDA memory, Numba labs, speedup analysis (next tutorial ii)

LLM fundamentals: tokens, attention, decoding, quantization

Deploying & using LLMs: local inference, OpenAI-compatible serving, remote APIs

Mini-Project: your dataset → EDA → meaningful analysis → launchpad proposal

HW4 Intro

Data analytics is not obsolete — it is now the data layer of AI infrastructure

Training data, RAG knowledge bases, eval sets, observability logs: all come from the pipelines you built in HW1–HW3

You must know the engine (GPU) and the model (LLM) before designing systems around them

- HW4 = build the skills (GPU + LLM), then Part 4 launches your Mini-Project
- Mini-Project = choose your own dataset and deliver a new, meaningful, impactful analysis

Textbook: S. Mitra, "System Design for the LLM Era", Publication Date: June 29, 2026.

Part 2 · LLM fundamentals (15 pts)

Q2.1 (10 pts): scaled dot-product attention by hand — Q, K, V length-3, d_k=2

Q2.2 (5 pts): tokens, decoding strategies, quantization, RAG & vector DBs (Ch. 1–2)

These definitions power every later task — the textbook builds on them in Ch. 1

Why is it faster? (and when is it not)

Amdahl’s law: speedup is capped by the serial fraction — transfer & launch overhead live there

Memory-bound kernels: limited by bandwidth (GB/s), e.g., vector add

Compute-bound kernels: limited by FLOPs, e.g., matrix multiply

Plot runtime vs N on log-log axes; expect GPU to win only above a crossover N

Bonus (Q1.4): tiled matmul with shared memory → report GFLOP/s; or torch.profiler; or Triton fused softmax

Scaled dot-product attention (Q2.1)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(Q \cdot K^T / \sqrt{d_k} \right) \cdot V$$

1. scores	$S = Q \cdot K^T$	(3×3 matrix)
2. scaled	$S' = S / \sqrt{d_k}$	$d_k = 2$
3. weights	$A = \text{row-softmax}(S')$	each row sums to 1
4. output	$O = A \cdot V$	(3×2 matrix)

No causal mask here — every position may attend to every position.

Show all intermediate matrices. Causal masking (lower-triangular) appears in decoder-only LLMs like Llama.

Tokens & decoding strategies

Token: the atomic unit the model consumes/produces; ~4 English chars, cost is billed per token

Greedy: always pick argmax → deterministic but repetitive

Temperature: scales logits before softmax; >1 more diverse, <1 more confident

top-k: sample only from the k most likely tokens; top-p (nucleus): sample from the smallest set with cumulative prob ≥ p

Deterministic vs. stochastic outputs matter in production (Ch. 1): grading needs determinism, chat needs diversity

Quantization: why 4-bit models are everywhere

fp16 7B model ≈ 14 GB → does not fit a T4; 4-bit ≈ 4 GB → fits

bitsandbytes: 8-bit (LLM.int8) and 4-bit NF4 — load any HF model quantized

GGUF: llama.cpp format, runs on CPU/GPU/Metal, single file, Ollama uses it

Trade-off: size / speed / some quality — measure all three in Q3.2

Ch. 2 cost optimization: smaller/cheaper model for easy tasks, big model for hard ones (model router)

Q3.1: deploy an LLM locally on Colab

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
model_id = "Qwen/Qwen2.5-1.5B-Instruct"   # or Llama-3.2-1B, Phi-3.5-mini
tok = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id, torch_dtype=torch.float16, device_map="auto")
inputs = tok("What is a large language model?", return_tensors="pt").to("cuda")
out = model.generate(**inputs, max_new_tokens=100,
                     temperature=0.7, top_p=0.9, do_sample=True)
print(tok.decode(out[0], skip_special_tokens=True))
```

Vary max_new_tokens / temperature / top_p / top_k and report VRAM with !nvidia-smi.

Q3.2: 4-bit loading with bitsandbytes

```
from transformers import BitsAndBytesConfig
nf4 = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16)
model = AutoModelForCausalLM.from_pretrained(
    model_id, quantization_config=nf4, device_map="auto")

# report: model size, VRAM (nvidia-smi), tokens/sec, answer quality
```

Compare fp16 / 8-bit / 4-bit / GGUF in one table. That table is your deployment decision evidence.

Q3.3: serve with an OpenAI-compatible endpoint

```
# llama.cpp (GGUF) or vLLM:
python -m llama_cpp.server \
  --model qwen2.5-1.5b-instruct-q4_k_m.gguf \
  --n_gpu_layers -1 --port 8000
curl http://localhost:8000/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{"model":"qwen","messages":[{"role":"user","content":"Hi!"}],
       "stream":false, "max_tokens":50}'
```

Measure TTFT and tokens/sec; try stream:true; then 4 concurrent requests and compare with a remote API.

Q3.3 (cont.): local vs. remote AI systems

	Local (Colab T4)	Remote API (e.g., GPT-4o-mini)
Setup	GGUF + llama.cpp / vLLM	API key + HTTP
Cost	Free GPU time	≈ $0.15 / 1M input tokens
Latency	TTFT ~0.2–2 s on T4	TTFT ~0.2–1 s
Control	Full: weights, cache, privacy	Prompt-level only
When to use	Offline, private data, high volume	Prototyping, peak load, big models

Q3.3 (4): design your model router (Ch. 2)

Route easy/cheap requests to the local small model; route hard/large requests to the remote model

Tiered fallback: local fails (OOM, crash) → retry remote; remote rate-limited → fall back to local

Circuit breaker stops calling a failing provider for a cool-down window

Prompt engineering (Q3.4)

Zero-shot: instruction only

Few-shot: add 2–5 labeled examples in the prompt → much better task adherence

Chain-of-thought: ask for step-by-step reasoning before the final answer

System prompt: set role, rules, output format once; keep user turns short

Deterministic output: ask for JSON, use JSON mode or function calling

Q3.4: classification with function calling

```
tools = [{"type": "function", "function": {
  "name": "classify_ticket",
  "description": "Classify a customer message",
  "parameters": {"type": "object",
    "properties": {"category": {"type": "string",
      "enum": ["refund", "shipping", "product", "other"]},
      "confidence": {"type": "number"}},
    "required": ["category", "confidence"]}}}]

# model returns tool_call: {"category": "refund", "confidence": 0.96}
```

Ch. 2 pattern: function calling (tool usage) → deterministic, schema-validated output for downstream code.

Part 4 · Mini-Project Launchpad (20 pts)

This semester's capstone: choose YOUR OWN dataset and deliver a new, meaningful, impactful analysis

Q4.1 (5 pts): dataset card (source, license, size) + 2–3 "so what?" questions + hypotheses

Q4.2 (7 pts): load (Spark / pandas) → profile → clean → 3+ EDA plots → initial insights

Q4.3 (8 pts): one substantive analysis + GPU/Spark speedup + LLM layer + 3 findings

Q4.4 (bonus 5): one-page Mini-Project proposal — dataset, question, method, metric, timeline

Q4.2: load → profile → clean (pandas / Spark)

```
# Q4.2: load → profile → clean (use Spark for big data)
import pandas as pd
df = pd.read_csv("your_dataset.csv")
df.info()                          # schema, dtypes, row count
print(df.isna().sum())             # missing values
print(df.duplicated().sum())       # duplicates
print(df.describe())               # outliers: min / max / quantiles
df = df.dropna(subset=["col"]).drop_duplicates()    # document each step
df["feature"] = df["a"] / df["b"]                   # engineered feature
df.groupby("cat")["y"].agg(["mean", "std"])         # first look at groups
```

Spark alternative: spark.read.csv(...) → df.printSchema() → groupBy().count(). Log every cleaning decision before/after.

Q4.3: meaningful analysis + GPU/LLM

One substantive analysis: trend / seasonality, anomaly detection, correlation / causality, prediction (train/test + metric), clustering

Accelerate one step: @cuda.jit kernel (Part 1) or Spark — report before/after runtime + speedup

LLM layer: executive summary, Q&A over your results, or mini-RAG over related text — note where it adds value vs. misleads

Three findings, each with evidence (table / plot / metric) + caveat + potential impact

HW4 → Mini-Project: the bridge

Mini-Project: choose your own dataset — stock / financial, IoT / sensor, or any domain you care about

The bar: a real question, a real analysis, an insight someone can act on — not a handful of plots

Carry over from HW4: dataset + EDA, accelerated analysis pipeline, LLM layer, proposal draft

Q4.4 bonus = your launchpad proposal draft

HW4 grading (100 pts)

Part	Content	Points
1	GPU programming (concepts, vector add, 2 labs, bonus)	35
2	LLM fundamentals (attention by hand, concepts)	15
3	Deploying & using LLMs (local, quantization, serving, prompts)	30
4	Mini-Project (dataset, EDA, analysis, launchpad proposal)	20
—	Total (+5 bonus, capped at 100)	100

Submission checklist

One PDF: UNI_HW4.pdf — code screenshots + explanations + results

Part 1 GPU programming: answers, plots, lab screenshots

Part 2 LLM fundamentals: hand-computed attention matrices, definitions

Part 3 Deploying and Using LLMs: generation outputs, quantization table, server logs, cost table

Part 4 Mini-Project: dataset card, cleaning log, EDA plots, analysis + speedup, LLM summary, 3 findings

Due: Thursday, Nov 5, 2026

Thank you!

Questions? Bring them to the Office Hours.