

System-on-Chip (SoC) Design

ECE382M.20, Fall 2026

Homework #1

Assigned: September 3, 2026

Due: September 17, 2026

Instructions:

- Please submit your solutions via Canvas. Submissions should include a single PDF with the writeup and single Zip or Tar archive for source code.
- You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions.

Problem 1: Edge LLM Execution (50 points)

Despite the popularity of large frontier LLMs (GPT-5, Claude Opus, Gemini 3, etc.), tiny LLMs can also perform well while remaining small enough to run on edge devices. One such model is Qwen2.5-0.5B, which has the following properties:

V: Vocabulary size = 151,936

N: Layers = 24

d: Hidden dimension = 896

H: Heads = 14

D_k : Head dimension = 64

Total parameters: 494,032,768 (~0.5 billion)

- LLM execution has two main phases: prefill and decode (i.e., generation). Assuming a batch size of one:
 - What are the differences between the prefill and decode phases of an LLM?
 - What type of vector or matrix operations (dot product/GEMV/GEMM) dominate at each phase?
- LLMs employ several algorithmic optimizations to ensure fast response times. One of these, used by Qwen2.5 and other models, is KV caching.
 - Why do LLMs use a KV cache during decode?
 - Assuming a context window size (L = prompt tokens + previously generated tokens) of 2048, a batch size of one, and standard FP16 (half) precision, what is the maximum KV cache size in bytes for an LLM with the parameters above (assume standard multi-headed attention)?
- Qwen2.5 includes a unique optimization: **grouped query attention**. Each attention layer in Qwen2.5 has 14 Query heads (H_Q) and 2 Key/Value Heads (H_{KV}).
 - Explain the difference between grouped query attention compared to traditional multi-headed attention.
 - How much memory does grouped-query attention save in the KV cache for Qwen2.5-0.5B?
- Model weights in LLMs can be stored at different quantization levels to reduce space overhead (but at the cost of some task performance).

- i) Describe or otherwise show how values are arranged in memory for the GGUF Q8_0 and Q4_0 quantization formats.
 - ii) How much space do the weights in Qwen2.5-0.5B require at FP32 (single precision), FP16 (half precision, the default), Q8_0 (GGUF 8-bit integer), and Q4_0 (GGUF 4-bit integer) quantization levels?
 - iii) Assuming our platform has 2 GB of DDR4 memory with 512 MB reserved for the OS and KV caches, which of these models will fit entirely in memory?
- e) For LLMs like Qwen2.5-0.5B, the generation phase is typically memory-bound and therefore limited by how fast we can stream weights from memory.
 - i) Assuming a maximum bandwidth of 9.6GB/s to the DDR4 memory, what is the upper bound on model speed (token/s) for each of the quantized Qwen2.5-0.5B models?
 - ii) In practice, you are unlikely to achieve this maximum theoretical bandwidth. Name one factor that may limit the maximum achievable bandwidth and therefore token generation speed.

Problem 2: LLM Kernels (50 points)

At the internal kernel level, much of the LLM arithmetic comes from linear projections that take an input activation vector and multiply it by a learned weight matrix. These projections appear in the attention portion of a transformer layer, where they create a query, key, and value vectors, and in the feed-forward portion of the layer, where they expand and transform the token representation.

A single projection for one token has the form

$$y = W x$$

Where $W \in R^{(M \times K)}$, $x \in R^K$, and $y \in R^M$. Here, x is the input activation for one token, W holds the learned weights for that layer, and y is the output activation passed to the next part of the network.

- a) Compute a small linear projection $y = Wx$ with

$$W = \begin{bmatrix} 1 & 3 & 6 \\ -2 & 2 & 4 \\ 0 & 5 & 3 \\ 2 & 3 & 0 \end{bmatrix} \quad x = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$$

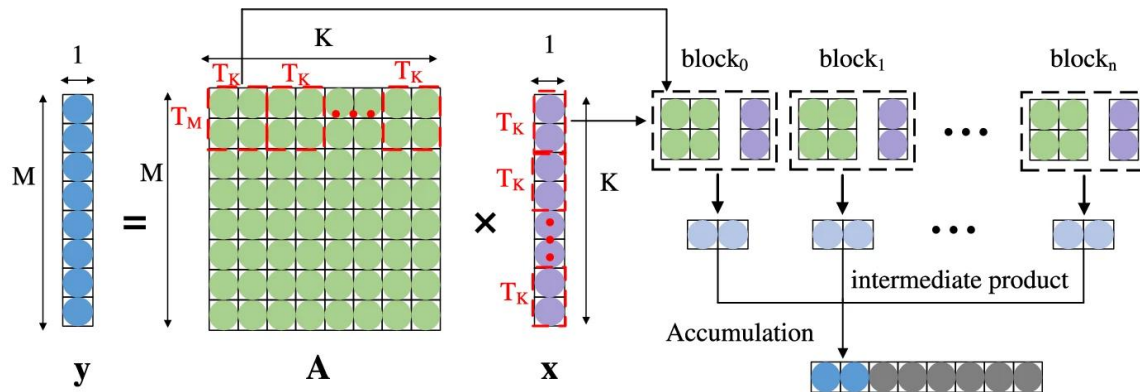
Show the dot product used to obtain each element of y .

- b) Express and implement the operation as a GEMV. For general M and K :
- i) Write straightforward pseudocode for computing $y = Wx$. Assume that W is stored in row-major order. (Do not use any GEMV library routines, i.e. write out the GEMV operation.)
 - ii) Give the exact number of scalar multiplications and additions performed by your pseudocode. Also give an equation for FLOP count, counting a multiply-add as two FP operations.

- iii) Describe the order in which your pseudocode accesses the elements of W and explain why that order is appropriate for the stated layout.
- c) Analyze data movement and arithmetic intensity. Assume W , x , and y contain 32-bit floating-point values. For this question, use the idealized assumption that one GEMV reads every element of W and x exactly once from main memory and writes every element of y exactly once to memory. Ignore cache-line effects and write allocation.
- Derive the total main memory traffic, in bytes, as a function of M and K .
 - Derive the arithmetic intensity in FLOPs per byte.
 - Evaluate the arithmetic intensity in the limit when both M and K are large.
 - Assuming a processor that has a peak compute throughput of P FLOP/s and memory bandwidth B bytes/s. Use a roofline-style argument to give an upper bound on GEMV performance.
- d) Optimize GEMV with blocking. A naive GEMV implementation will stream W and repeatedly access x , which does not make good use of cache or registers. In this part, we consider two levels of optimization. Assume that W is stored in row-major order.

At the first level, we try to keep the full input vector x in cache while streaming W and writing y row by row. The goal is to avoid repeatedly fetching x from main memory.

At the second level, we further tile computation to exploit register-level locality. Instead of processing one full row at a time, we process a tile of T_M output rows and T_K input columns as shown in the diagram below:



Source: <https://doi.org/10.1007/s11227-025-06993-6>

- For the first cache-level optimization, identify the minimum cache capacity C required as a function of M and K , assuming 32-bit data size. Describe any assumptions you have made. How does this type of caching change the number of main memory accesses and arithmetic intensity of the GEMV?
- For the second register-level optimization, identify the minimum number of registers needed in terms of T_M and T_K and explain your answer. How does this type of optimization change the number of main memory accesses/arithmetic intensity and the number of cache accesses that are made?