Assignment 2

Jose D. Pague

October 2025

1 Part A - Computer Architecture & Cache Fundamentals

1.1 What is a cache in memory systems?

1.1.1 Explain what a cache is, where it sits in the memory hierarchy, and why processors use it.

A cache is a small, high-speed memory found between the CPU and main memory that stores frequently accessed data or instructions. The processor uses the cached data/instructions to reduce latency and avoid fetching data that is frequently used from the slower DRAM. This improves performance on the CPU while keeping data frequently used closer to where computation is happening.

1.2 Purpose of a cache

1.2.1 Why do we add multiple cache levels (L1, L2, L3) instead of directly relying on main memory? What latency and bandwidth advantages does caching provide?

The purpose of a cache is to reduce the latency of main memory access. Each cache level is used to balance out speed, cost, and its overall capacity. L1 cache is closest to the main processor as it's the smallest and fastest found in each core (tens of KB). L2 is the second fastest, farther away with more caching capability (hundreds of KB), while L3 is further away but used by all cores and can handle more memory (tens of MB), and although slower than L2, it is still faster than DRAM. Caching provides adaptable latency and bandwidth advantages using this tiered hierarchy cache levels alternating between L1, L2, and L3 through direct need, which improves bandwidth, reduces latency, and overall system performance compared to relying solely on DRAM.

1.3 Cache hierarchy of a real device

1.3.1 Review a device of your choice (for example, your laptop, a Raspberry Pi, or a Jetson Orin Nano). Describe its memory hierarchy: registers, caches (L1/L2/L3), RAM type, and GPU memory if applicable. Provide available specifications (sizes, bandwidths, and latencies) from documentation or tools such as lscpu, nvidia-smi, or vendor datasheets.

The cache hierarchy of my desktop includes an Intel Core i7-12700K (12 cores / 20 threads) with a Base speed of 3.60 GHz and a cache hierarchy consisting of L1 cache (1.0 MB), L2 cache (12.0 MB), and L3 cache (25.0 MB). It features 32 GB of DDR4 RAM and an NVIDIA RTX 3060 Ti GPU with 8 GB of GDDR6 VRAM. The memory bandwidth is approximately 448 GB/s for the GPU and 50 GB/s for the CPU.

1.4 Cache hits and misses

1.4.1 Define cache hit and cache miss. Explain how each affects performance and how CPUs/GPUs try to minimize misses.

A cache hit and a cache miss are defined as data already in the cache, resulting in a successful request. A cache miss occurs when a request is not successful and the data is not found in the cache. It affects performance through increased latency as the data has to be fetched from the slower memory rather than from the cache itself. CPUs and GPUs reduce misses through techniques such as pre-fetching, larger associative caches, and optimization programs, such as with spatial and temporal locality.

1.5 Spatial and temporal locality

1.5.1 Explain these two principles and how they influence cache design. Give one program example that benefits from each.

Spatial locality refers to the surrounding nearby addresses that are commonly accessed at the same time due to close proximity. Temporal locality refers to data that has been recently used and is more than likely to be reused again soon. The usages in cache design are fetching whole blocks or lines instead of a single word, which increases the loaded data. This is used in program examples such as array iteration. Temporal locality involves accessing recently accessed lines in faster levels of the cache hierarchy for quick availability. A program example is a variable in a loop that's iterated each time and thus reused.

1.6 Advantages and limitations of caching

1.6.1 Summarize the main benefits of having caches in computing systems and mention one limitation or case where caching may not help.

The main benefits of computer systems having cache capability are reduced latency, rapid memory access, and overall improved performance, as the hierarchical cache structure allows for fast access to frequently used data. However, its limited capacity to predict irregular or random data access can reduce its effectiveness in certain workloads.

2 Part B - Theoretical & Analytical Questions on KV Cache

2.1 What is Auto Regressive Modeling in terms of Large Language Models (Transformers)? How do Transformers generate text in an auto-regressive fashion?

Auto Regressive Modeling in relation to LLMs (Transformers) refers to token prediction based on all previous tokens generated, one step at a time. Transformers achieve this through causal attention mechanisms, which prevent tokens from depending on future tokens and instead rely on past tokens for context. To maintain accuracy, multi-head attention improves its ability to capture relations from different parts of its input, which improves accuracy and remains consistent through its auto-regressive nature.

2.2 Explain in detail the concept of KV Cache in Large Language Models? Explain its purpose and advantages over standard implementation.

KV Cache in LLMs involves storing the Key and Value tensors generated from attention from previously created tokens, so they can be later used in the decoding process. Instead of recomputing each attention each time for every new token during the entire input sequence, the model instead calculates the new Key and Value for the current token and retrieves the prior ones from cache. This reduces redundant computation, improves inference speed, and memory efficiency. This leads to faster text generation and lower latency compared to the standard implementation.

2.3 KV-Cache Memory Footprint

2.3.1 Derive an expression for the total KV-cache size for one Transformer layer in bytes. Estimate the total cache size for your Assignment-1 model and verify if it fits in memory.

For each Transformer layer, the Key (K) and Value (V) tensors are stored for all previously generated tokens. B is the batch size, H_{kv} the number of attention heads, S the sequence length, d_{head} the head dimension, and b_{pe} the bytes per element (data type), then:

$$\text{KV bytes (per layer)} = B \times H_{kv} \times S \times d_{head} \times 2 \times b_{pe}$$

For the entire model with L layers:

$$\text{Total KV bytes} = L \times B \times H_{kv} \times S \times d_{head} \times 2 \times b_{pe}$$

2.4 Bandwidth vs Computation Bound

2.4.1 For one decoding step, which data movements dominate? Compare the theoretical memory bandwidth of your hardware to the data rate required to read/write to the KV cache at each step. Is decoding memory-bound or compute-bound?

The dominant data movement during one decoding step involves reading all previously stored Key and Value tensors for the current head:

$$\text{Read:} \;\approx 2 \times H_{kv} \times S \times d_{head} \times b_{pe}$$

and writing the new K and V tensors for the current token:

$$\text{Write:} \;\approx 2 \times H_{kv} \times d_{head} \times b_{pe}$$

Across L layers and batch size B:

$$\text{Bytes per token} \approx L \times B \times (2 \times H_{kv} \times S \times d_{head} \times b_{pe})$$

During decoding, the dominant cost is reading all previous K and V tensors from memory, while writes for new tokens are minimal. On my RTX 3060 Ti, the GPU bandwidth ($\approx 448 GB/s$) is much higher than the KV-cache data rate, making small models compute-bound. However, as sequence length or model size increases, the workload becomes increasingly memory-bound due to the growing cache read bandwidth requirements.

2.5 Latency Scaling with Sequence Length

2.5.1 Explain how complexity reduces from $O(S^2)$ without cache to $O(S)$ with cache. Describe how latency per token should change as S increases.

Without KV caching, each new token must recompute attention over all previously generated tokens, which leads to a quadratic complexity time of

$O(S^2)$. With caching, all previously computed Key and Value tensors are reused, so every new step will only attend to the stored history once, reducing its complexity to $O(S)$. Thus, latency per token increases more slowly and remains at a near constant rate as the sequence length increases, improving generation speed for long contexts.

2.6 Cache Placement and Architecture

2.6.1 Discuss how GPU/CPU cache hierarchy interacts with the KV cache. Which architectural features (tensor cores, shared memory, quantization) help mitigate bandwidth limits?

The KV cache found in GPU or CPU memory will have its efficiency dependent on the hardware cache hierarchy that manages data reuse during attention computation. GPU L1/L2 caches and shared memory buffer parts of the Key and Value tensors, which use spatial and temporal locality to reduce global memory access. The architectural features found in tensor cores for parallel matrix operations, shared memory for local reuse, and quantization techniques are used to shrink the KV tensor size to mitigate bandwidth limits and create an improvement in inference throughput.

2.7 Energy and Embedded Perspective

2.7.1 Why is KV-cache optimization important for energy-efficient edge inference? What are the trade-offs between cache precision (FP16 vs INT8) and power consumption?

KV-cache optimization is important for energy-efficient edge inference as it reduces redundant computations and memory accesses, allowing models to generate text faster and consume less power. Through reusable Key and Value tensors stored, inference latency and total energy output per token are greatly reduced instead of recomputing them through each step. Some tradeoffs between cache precision formats, such as FP16 or INT8, result in small accuracy losses while decreasing memory bandwidth and power consumption, which is both a pro and a con for achieving significant efficiency gains.

3 Part C — Coding and Experimental Tasks

3.1 Run a Pretrained Model

The pretrained **DistilGPT-2** model was loaded and verified by generating a short text prompt.

3.2 Experiment 1 — With and Without KV-Cache

Throughput, latency, and memory usage were measured with and without the key-value cache.

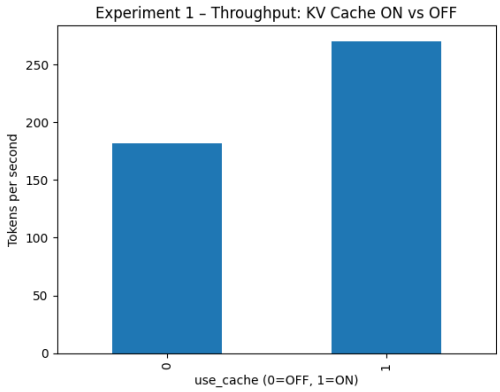

Figure 1: Throughput: KV-Cache ON vs OFF

3.3 Experiment 2 — Batching Effect

Tested batch sizes of 1, 2, and 4 prompts to observe changes in throughput and memory usage.

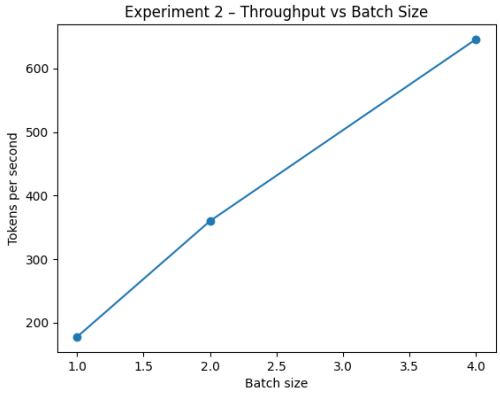

Figure 2: Throughput vs Batch Size

3.4 Experiment 3 — Cache Scaling with Sequence Length

Prompt lengths of 32, 128, and 256 tokens were used to measure latency and memory growth.

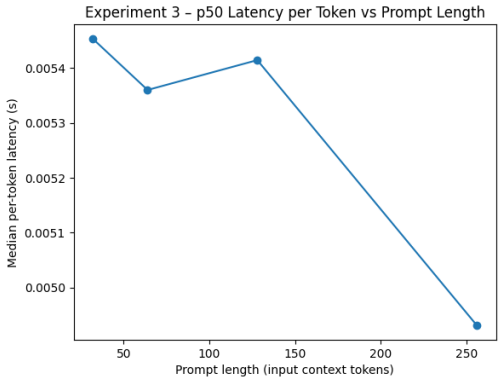

Figure 3: p50 Latency vs Prompt Length

4 Conclusion

The experiments revealed that the performance of Transformer inference was as expected. Through the usage of KV-cache, it improved throughput and reduced latency at a small cost of increased memory usage. Batching with multiple promoted improved GPU utilization and throughput, although it linearly increased memory consumption. Sequence length was shown to increase through cache memory at an approximate linear scale, while latency grew at a moderate rate. Thus, this aligns with the theoretical expectations for autoregressive generation in Transformer models.

To see GitHub Repository:

`https://github.com/josedavp/Assignment_2_EE5453_Jose.git`

The overall AI usage (reflection and raw queries), code, logs, outputs, README, and training screenshot can be found in the GitHub link.