# Lab 3: KV Cache Analysis in Small Language Models

**ECSE 397/600: Efficient Deep Learning**

Instructor: Prof. Gourav Datta

**Due Date: December 3, 11:59 PM EST**

---

## Objective

The purpose of this lab is to understand how **Key–Value (KV) caching** reduces computation and memory overhead during autoregressive generation in transformer-based language models. You will experiment with a lightweight model (`distilgpt2`) to:

- Measure the effect of cache size and sequence length on inference latency.
- Quantify the memory footprint of the KV cache.
- Visualize how latency scales with and without caching.

## Background

During autoregressive text generation, transformers repeatedly process all prior tokens at each decoding step, which scales as $O(L^2)$ in sequence length. The **KV cache** stores intermediate key and value tensors from past tokens so that only the new token's attention computation is performed, reducing the complexity to $O(L)$. However, this comes at the cost of additional memory usage, since the cache grows linearly with both sequence length and number of layers.

The memory per layer can be estimated as:

$$M_{KV} = 2 \times L \times D \times 2 \text{ bytes},$$

where $L$ is the sequence length, $D$ is the hidden dimension, and FP16 is assumed (2 bytes per element).

## Tasks

### Task 1: Load and Inspect Model

Use the Hugging Face Transformers library to load a small autoregressive model:

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch, time, psutil

model_name = "distilgpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name).to("cuda")
model.eval()
```

Inspect the model configuration (number of layers, hidden size, and attention heads) and record these values in your report.

### Task 2: Generation Without KV Cache

Write a function that generates tokens one by one, explicitly disabling caching. Measure average generation latency per token for sequence lengths $L = \{32, 64, 128, 256\}$.

```
def generate_no_cache(prompt, L):
    input_ids = tokenizer(prompt, return_tensors="pt").to("cuda")
    with torch.no_grad():
        for _ in range(L):
            outputs = model(**input_ids, use_cache=False)
            next_token = torch.argmax(outputs.logits[:, -1, :], dim=-1)
            input_ids = torch.cat([input_ids.input_ids, next_token.unsqueeze(-1)],
                dim=-1)
```

Record average latency (ms/token) and total memory usage using `psutil.virtual_memory()` or PyTorch's `torch.cuda.memory_allocated()`.

### Task 3: Generation With KV Cache Enabled

Repeat the same experiment but with caching enabled:

```
def generate_with_cache(prompt, L):
    input_ids = tokenizer(prompt, return_tensors="pt").to("cuda")
    past_key_values = None
    with torch.no_grad():
        for _ in range(L):
            outputs = model(**input_ids, use_cache=True, past_key_values=
                past_key_values)
            next_token = torch.argmax(outputs.logits[:, -1, :], dim=-1)
            past_key_values = outputs.past_key_values
            input_ids = next_token.unsqueeze(0)
```

Compare latency per token and memory footprint to the previous case. You should observe that caching provides a significant speed-up at the cost of a linearly increasing memory footprint.

### Task 4: Memory Estimation and Plot

Compute the theoretical KV cache memory per layer and total cache size across all layers using:

$$M_{\text{total}} = 2 \times L \times D \times N_L \times 2 \text{ bytes},$$

where $N_L$ is the number of layers and $D$ the hidden dimension. Plot both empirical and theoretical memory versus sequence length.

### Task 5: Optional (Bonus)

Quantize the KV cache tensors to 8-bit integers and measure new latency and memory usage. You may use:

```
past_key_values_8bit = tuple(
    tuple(kv.to(torch.uint8) for kv in layer)
    for layer in past_key_values
)
```

Discuss how such quantization might affect model accuracy or numerical stability.

## Deliverables

Submit a single `.zip` folder named `studentID_kv_cache_lab.zip` containing:

- Your Python notebook (`.ipynb`) or script file (`.py`).
- A short report (`report.pdf`) including:
    1. Model configuration (layers, hidden size, heads).
    2. Plots of latency vs. sequence length (with and without cache).
    3. Theoretical vs. measured KV cache memory plots.
    4. Discussion of trade-offs in latency and memory.

**References:** Hugging Face Documentation on Caching and Performance, Transformers GitHub Repository.