Assignment 3: Efficient LLM Inference

CS480/680: Introduction to Machine Learning

Designer: Arthur Chen | Instructor: Hongyang Zhang

Instructions

Submit three files to LEARN:

1. A PDF write-up containing answers to the written sub-questions (1.1 and 2.7).
2. A Python file named `<student_id>_assignment_llm.py` containing your completed functions and the provided `main()`.
3. A JSON artifact file named `<student_id>_artifact.json`, produced by `python <student_id>_assignment_llm.py -output <student_id>_artifact.json`.

All coding exercises live in the companion notebook `assignment_llm.ipynb`. Complete the notebook, copy your implementations into `<student_id>_assignment_llm.py`, then generate the JSON artifact. We run automated scripts to check both:

(1) the public artifact values generated by your script, and

(2) the individual functions on hidden test cases.

Rules:

- *We do not accept* hand-written submissions – only the above listed files will be graded.
- Name your files `<student_id>_assignment_llm.pdf`, `<student_id>_assignment_llm.py`, and `<student_id>_artifact.json`. *Incorrect naming may result in a grade of 0.*
- Use only the Python standard library and NumPy. Do *not* use PyTorch, TensorFlow, JAX, HuggingFace, or external LLM libraries.
- Floating-point answers are checked with tolerance 10^{-6} unless otherwise stated.
- *Do not change* the names, inputs, or outputs of the required functions (including keyword arguments and their default values).
- This assignment is due on Aug 5th, 2026 at 11:59 AM ET (*noon*).

Question 1: GPT-2, Autoregressive Decoding, and KV Cache — 12 points

GPT-2 is an autoregressive transformer language model. Given tokens $x_1, \ldots, x_T$, it models

$$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_{<t}).$$

In this question you implement small components that reflect GPT-2-style autoregressive inference.

1.1 Autoregressive Language Modeling — Written · 2 pts

Answer briefly in your PDF:

(a) (1 point) Why is GPT-2 called an *autoregressive* language model?

(b) (1 point) Why does GPT-2 use a causal attention mask during training and inference?

1.2 Sliding-Window Causal Mask — Coding · 2 pts

Efficient transformers such as Mistral and Longformer restrict each token to a local window of recent tokens. Implement (notebook Section 1.2):

```
make_causal_mask(T, window=None)
```

returning an integer mask with

$$\text{mask}[i, j] = 1 \quad \text{iff} \quad j \le i \;\; \textbf{and} \;\; (\texttt{window is None} \;\text{ or }\; i - j < \text{window}).$$

With `window=None` this is the standard causal mask. For example:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\texttt{make\_causal\_mask(4)}} \qquad \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}}_{\texttt{make\_causal\_mask(5, window=2)}}$$

1.3 Shifted Language Modeling Loss with Padding — Coding · 3 pts

The logits at position t predict the next token x_{t+1}. Padded targets must not contribute to the loss, so (as in PyTorch's `cross_entropy`) any position whose *target* equals `ignore_index` is excluded from both the sum and the average. Implement (notebook Section 1.3):

```
shifted_lm_cross_entropy(logits, tokens, ignore_index=-100)
```

$$\text{loss} = -\frac{1}{|\mathcal{V}|} \sum_{t \in \mathcal{V}} \log \text{softmax}(\text{logits}[t])\,[\text{tokens}[t+1]],$$

$$\text{where} \quad \mathcal{V} = \{\, t \in \{0, \ldots, T-2\} : \text{tokens}[t+1] \neq \texttt{ignore\_index} \,\}.$$

Ignore `logits[T-1]` (no next token). If every target is ignored, return `0.0`.

1.4 Multi-Head Single-Step Attention with a KV Cache

Coding · 3 pts

During autoregressive inference, previously computed keys and values are stored in a KV cache. Implement *multi-head* single-step attention (notebook Section 1.4):

```
cached_attention_step(q, K_cache, V_cache, num_heads)
```

With $D = \text{num\_heads} \cdot \text{head\_dim}$, head h occupies columns $[h \cdot \text{head\_dim}, (h{+}1) \cdot \text{head\_dim})$ and attends independently, scaled by $\sqrt{\text{head\_dim}}$ (*not* $\sqrt{D}$):

$$\text{scores}_i^{(h)} = \frac{q^{(h)} \cdot K_{\text{cache}}^{(h)}[i]}{\sqrt{\text{head\_dim}}}, \quad w^{(h)} = \text{softmax}(\text{scores}^{(h)}), \quad \text{output}^{(h)} = \sum_i w_i^{(h)} V_{\text{cache}}^{(h)}[i].$$

Concatenate the per-head outputs into a vector of length D; return the weights as a (num_heads, t) array.

1.5 KV Cache Memory with Grouped-Query Attention

Coding · 2 pts

Under Grouped-Query Attention (GQA), many query heads share a smaller number of key/value heads, so the cache is sized by `num_kv_heads` (the groups), not the number of query heads. Implement (notebook Section 1.5):

```
kv_cache_bytes(num_layers, batch_size, seq_len, num_kv_heads, head_dim,
bytes_per_value)
```

$\text{total\_bytes} = 2 \cdot \text{num\_layers} \cdot \text{batch\_size} \cdot \text{seq\_len} \cdot \text{num\_kv\_heads} \cdot \text{head\_dim} \cdot \text{bytes\_per\_value}.$

The factor of 2 accounts for storing both keys and values.

Question 2: Speculative Decoding and Efficient LLM Serving

13 points

Speculative decoding accelerates LLM inference by using a small draft model p_D to propose tokens and a large target model p_T to verify them. At a verification position, if the draft proposes token y, the acceptance probability is

$$A(y) = \min\left(1, \frac{p_T(y)}{p_D(y)}\right).$$

2.1 Acceptance Probabilities

Coding · 1 pts

Implement (notebook Section 2.1):

```
acceptance_probabilities(p_d, p_t)
```

returning $\text{accept\_probs}[y] = \min(1, p_t[y]/p_d[y])$, assuming $p_d[y] > 0$ for all y.

2.2 Exact Expected Acceptance Rate

Coding · 2 pts

Implement (notebook Section 2.2):

```
exact_acceptance_rate(p_d, p_t)
```

$$\text{rate} = \mathbb{E}_{y\sim p_d}\left[\min\left(1, \frac{p_T(y)}{p_D(y)}\right)\right] = \sum_y p_D(y)\cdot\min\left(1, \frac{p_T(y)}{p_D(y)}\right).$$

2.3 Deterministic Sampling Helper

Coding · 1 pts

To make grading deterministic, implement inverse-CDF sampling from a provided uniform $u \in [0, 1]$ (notebook Section 2.3):

```
sample_from_distribution(probs, u)
```

Return the smallest index i such that cumsum(probs)$[i] > u$; if no such index exists (e.g. $u = 1$, or floating-point round-off leaves the final cumulative sum just below u), return the last index $V - 1$, where $V =$ `len(probs)`. For example, `probs`$= [0.2, 0.5, 0.3]$ with $u = 0.6$ has cumulative sums $[0.2, 0.7, 1.0]$ and returns `1`.

2.4 One Round of Speculative Decoding

Coding · 4 pts

Implement one round of speculative decoding (notebook Section 2.4):

```
speculative_decode_round(draft_tokens, p_d_list, p_t_list, accept_uniforms,
sample_uniforms)
```

The draft model has proposed K tokens $y_1, \ldots, y_K$. For each drafted token y_i, distributions $p_D^{(i)}$ and $p_T^{(i)}$ are given. The target model also provides one extra distribution $p_T^{(K+1)}$, used if all drafted tokens are accepted. See the notebook for the full algorithm, including residual sampling on rejection.

2.5 Serving-Level Speedup Simulation

Coding · 1 pts

Each target-model verification pass can generate multiple tokens. Implement (notebook Section 2.5):

```
simulate_target_passes(max_new_tokens, accepted_counts)
```

In each round, tokens_generated = accepted_count + 1, and `accepted_counts` is cycled until at least `max_new_tokens` tokens are produced.

2.6 Expected Tokens per Round

Coding · 2 pts

Drafted tokens are verified in order, so the first i are all accepted with probability $\prod_{j=1}^{i} a_j$, where $a_1, \ldots, a_K$ are the per-position acceptance probabilities. Since one extra token is always emitted, implement (notebook Section 2.6):

```
expected_tokens_per_round(accept_probs)
```

$$\mathbb{E}[\text{tokens per round}] = 1 + \sum_{i=1}^{K}\prod_{j=1}^{i} a_j.$$

2.7 Acceptance, Total Variation, and Correctness

Written · 2 pts

Answer in your PDF:

(a) (1 point) Show that the expected acceptance rate equals

$$\sum_y p_D(y) \min\left(1, \frac{p_T(y)}{p_D(y)}\right) = \sum_y \min\left(p_D(y), p_T(y)\right) = 1 - \mathrm{TV}(p_D, p_T),$$

where $\mathrm{TV}(p_D, p_T) = \frac{1}{2}\sum_y |p_D(y) - p_T(y)|$. *(Hint:* $\min(a, b) = \frac{1}{2}(a + b - |a - b|)$.*)*

(b) (1 point) Why does speculative decoding preserve the target model distribution while reducing the number of target-model forward passes?

Point Summary

Part	Component	Type	Points
1.1	Autoregressive LM explanation	Written	2
1.2	Sliding-window causal mask	Coding	2
1.3	Shifted LM loss with padding	Coding	3
1.4	Multi-head cached attention step	Coding	3
1.5	GQA KV cache memory	Coding	2
	Question 1 subtotal		**12**
2.1	Acceptance probabilities	Coding	1
2.2	Exact acceptance rate	Coding	2
2.3	Deterministic sampling helper	Coding	1
2.4	Speculative decoding round	Coding	4
2.5	Serving speedup simulation	Coding	1
2.6	Expected tokens per round	Coding	2
2.7	Acceptance / TV & correctness	Written	2
	Question 2 subtotal		**13**
	Total		**25**