CS726: Programming Assignment 3

Total Points: 50

March 29, 2025

General Instructions

1. Plagiarism will be strictly penalized including but not limited to reporting to DADAC and zero in assignments. If you use tools such as ChatGPT, Copilot, you must explicitly acknowledge their usage in your report. While limited use of such tools is permitted, relying on them for the entire assignment will lead to penalties. Additionally, if you use external sources (e.g., tutorials, papers, or open-source code), you must cite them properly in your report and comments in the code.

2. Submit a report explaining your approach, implementation details, results, and findings. Clearly mention the contributions of each team member in the report. Submit your code and report as a compressed `<TeamName>_<student1rollno>_<student2rollno>_<student3rollno>.zip` file. Fill a student roll number as `NOPE` if less than 3 members.

3. Start well ahead of the deadline. Submissions up to two days late will be capped at 80% of the total marks, and no marks will be awarded beyond that.

4. Do not modify the environment provided. Any runtime errors during evaluations will result in zero marks. `README.md` provides instructions and tips to set up the environment and run the code.

5. Throughout the assignment, you have to just fill in your code in already existing files. Apart from the report, do not submit any additional models or files. The internal directory structure of your final submission should look as follows:

```
cs726_assgmt3/
  |
  +- task0.py
  +- generate.py [to be changed]
  +- task1.py
  +- generate_constrained.py [to be changed]
  +- word_lists.txt
  +- task2.py
  +- generate_medusa.py [to be changed]
  +- README.md
  +- PA3_Problem_Statement.pdf
  +- report.pdf [NEW]
```

6. STRICTLY FOLLOW THE SUBMISSION GUIDELINES. Any deviation from these guidelines will result in penalties.

1 Problem Statement

1.1 Task 0: Introduction to LLM Decoding Techniques [15 points]

This section is designed to get you familiar with the decoding process in Large Language Models (LLMs) and how different sampling techniques impact its text generation. Your task is to implement and analyze the following decoding strategies on Llama-2 [3] when evaluated on Hindi to English translation task with IN22-Gen [2] dataset using relevant metrics (details about the evaluation metrics are provided at the end of the section).

(a) **Greedy Decoding:** At every step, you simply pick the token with the highest probability from the LLM's output distribution. Formally, at the t^{th} step, you obtain the next token as follows:

$$y_t = \arg\max_w P(w \mid y_{1:t-1}, x)$$

where $y_{1:t-1}$ denotes previously generated tokens and x is the input prompt. This process is repeated iteratively until the end-of-sentence (EOS) token is generated. [3 pts]

(b) **Random Sampling with Temperature Scaling:** Instead of always selecting the most probable token, here we randomly sample from the probability distribution while adjusting its sharpness using a temperature parameter τ. That is, first, we modify the probabilities as follows:

$$P'(w \mid y_{1:t-1}, x) = \frac{P(w \mid y_{1:t-1}, x)^{1/\tau}}{\sum_{w' \in V} P(w' \mid y_{1:t-1}, x)^{1/\tau}}$$

A token is then randomly sampled from P'. Like before, keep repeating this process until the EOS token is generated. Here, you must experiment with $\tau \in \{0.5, 0.9\}$ and report your findings. [3 pts]

(c) **Top-k Sampling:** Rather than sampling from the entire vocabulary, in Top-k sampling, we restrict our choices to the k most probable tokens. To do this, first, we sort the vocabulary by probability and keep only the top k tokens:

$$V_k = \{w_1, w_2, ..., w_k\}, \quad \text{where } P(w_i) \geq P(w_{i+1}) \text{ for } i < k$$

The probabilities within V_k are then normalized as follows:

$$P'(w) = \begin{cases} \frac{P(w)}{\sum_{w' \in V_k} P(w')} & \text{if } w \in V_k \\ 0 & \text{otherwise} \end{cases}$$

A token is then randomly sampled from P'. As before, repeat the process until the EOS token is generated. Here, experiment with $k \in \{5, 10\}$ and report your findings. [4 pts]

(d) **Nucleus Sampling:** Instead of picking a fixed number of tokens as in Top-k Sampling, here we dynamically choose the smallest set of tokens whose cumulative probability exceeds a threshold p:

$$V_p = \{w_1, w_2, ..., w_m\}, \quad \text{such that} \quad \sum_{i=1}^{m} P(w_i) \geq p$$

We then normalize probabilities over this set and sampling occurs as follows:

$$P'(w) = \begin{cases} \frac{P(w)}{\sum_{w' \in V_p} P(w')} & \text{if } w \in V_p \\ 0 & \text{otherwise} \end{cases}$$

Similar to before, repeat the process until the EOS token is generated. Here, experiment with $p \in \{0.5, 0.9\}$ and report your findings. [5 pts]

For each decoding technique, generate text outputs and evaluate them using the following metrics:

(a) **BLEU Score:** Measures the similarity between generated and reference text based on n-gram overlap.

(b) **ROUGE Score:** Measures the overlap of n-grams, sequences, and longest common subsequences between generated and reference text.

More details about these metrics can be found here.

1.2 Task 1: Word-Constrained Decoding [15 pts]

In this section, we will implement a variant of Grammar-Constrained Decoding called Word-Constrained Decoding. Assume that there is an oracle that has magically provided you, for every test example, a bag of all the words that appear in its output. Your task is to design a greedy decoding technique that takes advantage of this additional word list and improve the LLM performance. Keep in mind that the LLM may tokenize a word as a single token or a sequence of tokens. Similar to Section 1.1, you will report BLEU and ROUGE scores and compare this technique against the strategies explored in Section 1.1. [*Hint: Trie*]

1.3 Task 2: Staring into Medusa's Heads [20 pts]

In this section, we will explore a speculative decoding framework called MEDUSA [1]. Figure 1 provides an illustration of MEDUSA's architecture. The core idea behind MEDUSA is straightforward: in addition to the standard Language Modeling (LM) head, you also train multiple MEDUSA heads (a.k.a linear layers) that operate on the final hidden states of the LLM and are responsible for predicting some token in the future. Specifically, if $y_{1:t-1}$ is the input sequence to the LLM, then the LM head predicts token y_t, while the first MEDUSA head predicts token y_{t+1}, the second head predicts y_{t+2} and so on, until the K^{th} decoding head predicts y_{t+K} (i.e., $(K+1)^{\text{th}}$ token in the future). By strategically utilizing these decoding heads, MEDUSA can predict several subsequent tokens in parallel, enabling faster inference compared to the traditional auto-regressive decoding. Before proceeding, we strongly recommend that you go through the original paper. However, you may skip details related to TREE ATTENTION and other aspects of the original inference strategy, as we will be implementing a simpler technique in this section.

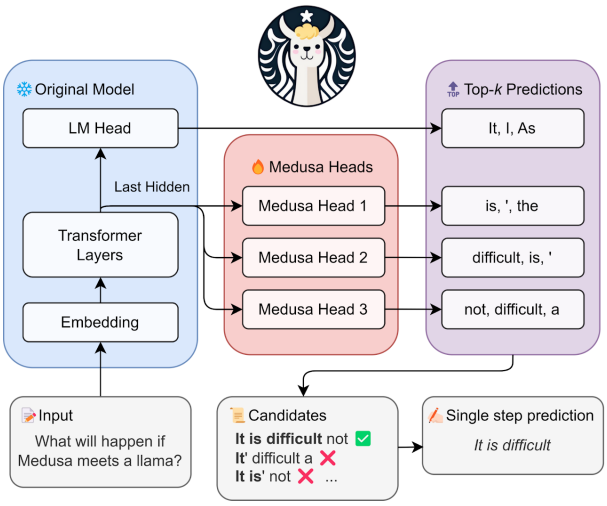

Figure 1: MEDUSA's architecture

The original inference strategy proposed in the paper is fairly complex and involves coding up complicated mechanisms. Therefore, in this section, we will explore simpler alternative decoding strategies (NOTE: Do not be alarmed if the predictions are worse than previous settings). Specifically, your task is to implement the following two approaches:

(a) **Single Head Decoding:** In this approach, we use only the LM head to perform inference. That is, at each step, we greedily pick the most probable token from the LM head's output distribution. This predicted token is then fed back as input to the LLM, and the cycle repeats. [5 pts]

(b) **Multi Head Decoding:** Here, we will utilize MEDUSA's decoding heads along with the existing LM head to generate multiple future tokens simultaneously. Let us consider $y_{1:t-1}$ as the input to the LLM and K be the total number of MEDUSA heads available, out of which we want to use first S heads. Then the decoding strategy works as follows:

(i) `STEP 1`: First, obtain the logits $\{p_t, p_{t+1}, \dots p_{t+S}\}$ by passing $y_{1:t-1}$ as input to the LLM. Here, p_t denotes the logit from the LM head, while p_{t+k} corresponds to the logit from the k^{th} MEDUSA head.

(ii) `STEP 2`: Perform beam search (with a beam width of W) over these $S+1$ logits to generate W candidate sequences, denoted as `candidates` $= \{\hat{y}^1_{1:t+S}, \hat{y}^2_{1:t+S}, \dots \hat{y}^W_{1:t+S}\}$. Note here that each candidate has $S+1$ new tokens added to $y_{1:t-1}$ and there are W such candidates. Algorithm 1 gives a detailed overview of the beam-search algorithm.

Algorithm 1: Beam Search

Input: $\{p_t, p_{t+1}, \dots, p_{t+S}\}, y_{1:t-1}$
Output: $\{\hat{y}^1_{1:t+S}, \hat{y}^2_{1:t+S}, \dots \hat{y}^W_{1:t+S}\}$

```
candidates ← {y_{1:t-1}}
scores ← {0.0}
// Loop over all the probability distributions and keep track of valid
   candidates as you progress.
for s = 0 to S do
    logp_{t+s} ← log_softmax(p_{t+s})
    newCandidates ← {}
    newScores ← {}
    // For each candidate in the beam, we extend it by 1 token and
       calculate the new score.  This score is then used to identify
       Top-W new candidates.
    for c = 1 to len(candidates) do
        for ŷ ∈ TopW(logp_{t+s}) do
            new_score ← scores[c] + logp_{t+s}[ŷ]
            new_candidate ← candidates[c] ⋃ {ŷ}
            newScores.append(new_score)
            newCandidates.append(new_candidate)
        candidates ← TopW(newCandidates, newScores)
    // Retain only top W candidates.  Here, newScores is used to find
       those Top W entries.
    candidates, scores ←TopW(newCandidates, newScores)
return candidates;
```

(iii) `STEP 3`: Finally, use the LM head to compute scores for all candidate sequences and pick the one with the highest score. Specifically, for a candidate sequence $\{\hat{y}_1, \dots, \hat{y}_{t-1}, \hat{y}_t, \hat{y}_{t+1}, \dots \hat{y}_{t+S}\}$, the score can be computed as:

$$\texttt{Score} = \sum_{i=t}^{t+S} logp_i[\hat{y}_i]$$

where $logp_i \leftarrow \text{log\_softmax}(p_i)$.

Repeat the above steps until you encounter an EOS token. Similar to before, you will report BLEU and ROUGE scores, along with Real Time Factor (RTF) for $W \in \{2, 5, 10\}$ and number of medusa heads $S \in \{2, 5\}$ and report your findings. [15 pts]

Additional Resources

- Illustration of the Transformer Architecture by Jay Alammar
- Building GPT from Scratch by Andrej Karpathy
- Chapter 9, 10, 11, 12 of Speech and Language Processing by Dan Jurafsky and James H. Martin

References

[1] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024.

[2] Jay Gala, Pranjal A Chitale, A K Raghavan, Varun Gumma, Sumanth Doddapaneni, Aswanth Kumar M, Janki Atul Nawale, Anupama Sujatha, Ratish Puduppully, et al. Indictrans2: Towards high-quality and accessible machine translation models for all 22 scheduled indian languages. *Transactions on Machine Learning Research*, 2023.

[3] Hugo Touvron, Louis Martin, et al. Llama 2: Open foundation and fine-tuned chat models, 2023.