CS726 Programming Assignment 3
LLM Sampling and Decoding Techniques

Kshitij Vaidya
Raunak Mukherjee
Anshu Arora

March 31, 2025

Contents

1 Task 0 : Introduction to LLM Decoding

To familiarise ourselves with the decoding techniques in Large Language Models, we implemented functions to translate Hindi to English using the Llama-2 model and the data was used from the IN22-Gen dataset. The **BLEU** and **ROUGE** scores were used for evaluation of the translations.

1.1 Decoding Techniques

1. **Greedy Decoding** : Here, at every step we select the token with the highest probability from the output distribution of the LLM. Mathematically, we represent this as :

$$\hat{y}_t = \arg\max_{y_t} P(y_t|y_{<t}, x) \tag{1}$$

   where $\hat{y}_t$ is the token selected at time step t and $y_{<t}$ is the sequence of tokens selected till time step $t-1$. This sequence repeats till we reach an EOS token or the maximum output length of the sequence.

2. **Random Sampling with Temperature** : Here, we randomly sample from the probability distribution of the output tokens. The sharpness of the distribution is controlled by the temperature parameter τ. We first modify the distribution as :

$$P'(w|y_{<t}, x) = \frac{P(w|y_{<t}, x)^{1/\tau}}{\sum_{w'} P(w'|y_{<t}, x)^{1/\tau}} \tag{2}$$

   and then sample from this distribution. Here, higher values of τ lead to a more uniform distribution and lower values lead to a sharper distribution. Thus, lower values of τ lead to more deterministic sampling and perform similar to Greedy Decoding.

3. **Top-K Sampling** : We restrict our choices to the K most probable tokens in the distribution. We first sort the tokens in decreasing order of probability and then sample from the top-K tokens. This is mathematically represented as :

$$V_k = \{w_1, w_2 \cdots w_k\}, where P(w_i) \geq P(w_{i+1}) for i < k \tag{3}$$

   The probabilities are then normalised and we select the token by sampling from this distribution

4. **Nucleus Sampling** : We take the smallest set of tokens whose cumulative probability is greater than a threshold p. We first sort the tokens in decreasing order of probability and then select the smallest set of tokens whose cumulative probability is greater than p. This is mathematically represented as :

$$V_p = \{w| \sum_{w' \in V_p} P(w') \geq p\} \tag{4}$$

   The probabilities are then normalised and we select the token by sampling from this distribution

1.2 Evaluation Metrics

1. **BLEU Score** : The BLEU score is a metric used to evaluate the quality of machine-translated text. It is based on the n-gram precision of the translated text. The BLEU score is calculated as :

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{5}$$

where p_n is the n-gram precision and w_n is the weight assigned to the n-gram precision. The brevity penalty BP is used to penalise shorter translations.

2. **ROUGE Score** : The ROUGE score is a metric used to evaluate the quality of summaries. It is based on the overlap of n-grams between the generated summary and the reference summary. The ROUGE score is calculated as :

$$\text{ROUGE} = \frac{\sum_{n=1}^{N} \text{ROUGE}_n}{N} \tag{6}$$

where ROUGE_n is the n-gram overlap between the generated summary and the reference summary.

1.3 Results

Method	BLEU	ROUGE-1	ROUGE-2	ROUGE-LCS
Greedy	0.3099	0.3543	0.1296	0.2710
Random ($\tau = 0.5$)	0.2928	0.2932	0.0931	0.2266
Random ($\tau = 0.9$)	0.1985	0.2000	0.0585	0.1647
Top-k ($k = 5$)	0.2476	0.2464	0.0698	0.1826
Top-k ($k = 10$)	0.2490	0.2584	0.0797	0.1984
Nucleus ($p = 0.9$)	0.0846	0.1473	0.0422	0.1153

Table 1: Comparison of different decoding methods using BLEU and ROUGE scores

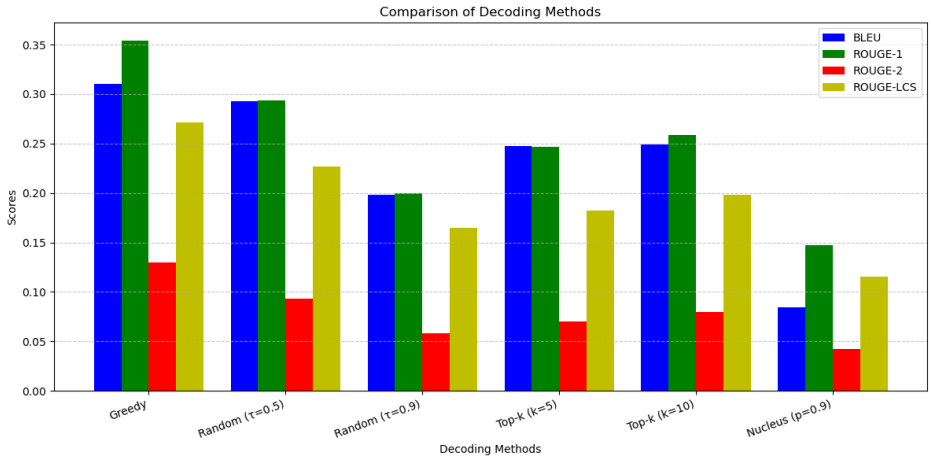

Figure 1: Comparison of different decoding methods using BLEU and ROUGE scores

2 Task 1 : Word-Constrained Decoding

Here, a variant of Grammar-Constrained Decoding is used which is called Word Constrained Decoding. In this, we are given a set of words that must appear in the output sequence. We implement a greedy decoding technique that takes this set of words in account improves the LLM performance. We use a Trie based implementation to store the set of words and use it to check if the word is present in the output sequence. The algorithm is as follows :

1. We build the Trie using the set of words that must appear in the output sequence.
2. We then initialise the output sequence as an empty list and the current word as an empty string.
3. We then iterate over the output sequence and check if the current word is present in the Trie. If it is present, we add the current word to the output sequence and reset the current word to an empty string.
4. If the current word is not present in the Trie, we add the current token to the current word and continue.
5. We repeat this process till we reach the end of the output sequence.

2.1 Trie Implementation

The following Python code implements a Trie data structure to manage a word list. The Trie efficiently supports insertion, prefix checking, and word completion validation.

```
from collections import defaultdict

class TrieNode:
    def __init__(self):
        self.children = defaultdict(TrieNode)
        self.isEndOfWord = False

```

```
class Trie:
    def __init__(self, tokenizer):
        self.root = TrieNode()
        self.tokenizer = tokenizer

    def insert(self, word):
        tokens = self.tokenizer.encode(word, add_special_tokens=False)
        node = self.root
        for token in tokens:
            node = node.children[token]
        node.isEndOfWord = True

    def startsWith(self, prefixTokens):
        node = self.root
        for token in prefixTokens:
            if token not in node.children:
                return False
            node = node.children[token]
        return True

    def validCompletion(self, prefixTokens) -> bool:
        node = self.root
        for token in prefixTokens:
            if token not in node.children:
                return False
            node = node.children[token]
        return node.isEndOfWord
```

Listing 1: Trie Implementation

2.2 Results

The word constrained decoding technique and the greedy decoding technique were used to translate Hindi to English. The BLEU and ROUGE scores were used for evaluation of the translations. The results show that this method performs better then the greedy decoding technique marginally. The BLEU and ROUGE scores are as follows :

Metric	**Score**
BLEU	0.19298
ROUGE-1	0.43248
ROUGE-2	0.27158
ROUGE-LCS	0.39884

Table 2: Task 1 - BLEU and ROUGE Scores

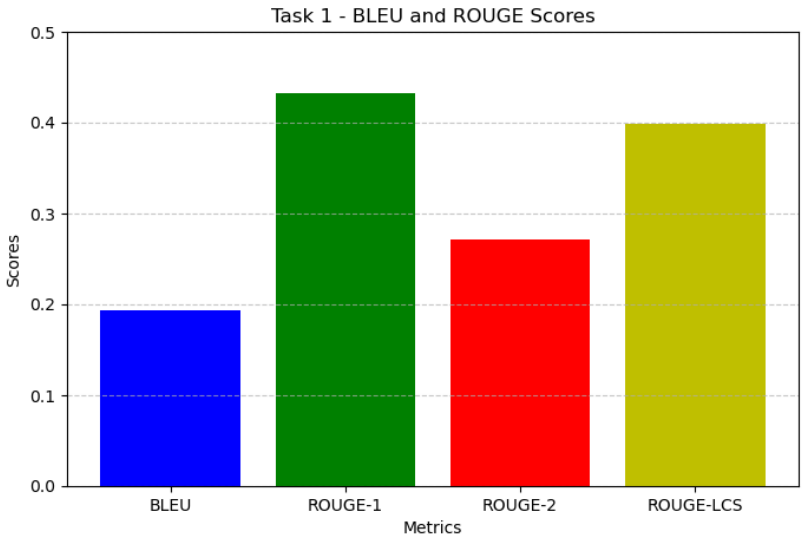

Figure 2: Task 1 - BLEU and ROUGE Scores

3 Task 2 : Staring into Medusa's Heads

Speculative decoding is a technique designed to accelerate the inference process of Large Language Models (LLMs) by leveraging additional predictive heads to generate multiple future tokens in parallel. The Medusa architecture introduces multiple auxiliary heads, referred to as Medusa heads, which work alongside the standard Language Modeling (LM) head to predict tokens at different future time steps. This enables faster text generation by reducing the number of sequential steps required in the decoding process.

3.1 Medusa Architecture

The core idea behind Medusa is to introduce multiple linear layers, known as Medusa heads, that operate on the final hidden states of the LLM. Given an input sequence $y_{1:t-1}$:

- The LM head predicts the next token y_t.
- The first Medusa head predicts y_{t+1}.
- The second Medusa head predicts y_{t+2}.
- This continues up to the Kth Medusa head, which predicts the $(t+K)$th token.

By leveraging these additional predictions, speculative decoding enables multiple tokens to be generated in parallel, leading to a significant speedup over traditional autoregressive decoding.

3.2 Decoding Strategies

We explore two different decoding strategies based on Medusa:

3.2.1 Single Head Decoding

In this approach, only the LM head is used for inference. At each step, the model greedily selects the most probable token from the LM head's output distribution and feeds it back as input to the LLM. This process repeats until the sequence is fully generated.

3.2.2 Multi Head Decoding

This approach utilizes Medusa heads to generate multiple future tokens in a single decoding step. The process consists of the following steps:

1. **Step 1: Compute Probability Distributions**
   Given an input sequence $y_{1:t-1}$, the LLM computes the probability distributions for the next $S+1$ tokens:
   $$\{p_t, p_{t+1}, ..., p_{t+S}\} \tag{7}$$
   where p_t is predicted by the LM head, and p_{t+k} is predicted by the kth Medusa head.

2. **Step 2: Beam Search over Future Tokens**
   Beam search with beam width W is applied over these probability distributions to generate W candidate sequences. Each candidate sequence contains $S+1$ newly predicted tokens.

3. **Step 3: Score Computation and Selection**
   The LM head is used to compute scores for all candidate sequences. The final sequence is selected based on the highest cumulative log probability:
   $$\text{Score} = \sum_{i=t}^{t+S} \log p_i[y_i] \tag{8}$$
   This process is repeated until an End-of-Sequence (EOS) token is encountered.

3.3 Performance Evaluation

To assess the effectiveness of speculative decoding, we evaluate BLEU and ROUGE scores along with the Real Time Factor (RTF) for different values of beam width ($W \in \{2, 5, 10\}$) and the number of Medusa heads ($S \in \{2, 5\}$). The results highlight the trade-offs between inference speed and output quality, as increasing S leads to faster generation but may impact accuracy.

Method	BLEU	ROUGE-1	ROUGE-2	ROUGE-LCS	RTF
Single Head (Greedy)	0.2881	0.3994	0.1471	0.3166	0.1059
Multi Head (W=2, S=2)	0.2091	0.3213	0.0898	0.2530	0.1944
Multi Head (W=2, S=5)	0.2564	0.3700	0.1283	0.2958	0.2615
Multi Head (W=5, S=2)	0.1593	0.2811	0.0652	0.2161	0.3711
Multi Head (W=5, S=5)	0.2150	0.3461	0.1083	0.2783	0.3141

Table 3: Comparison of Single Head and Multi Head Decoding using BLEU and ROUGE scores

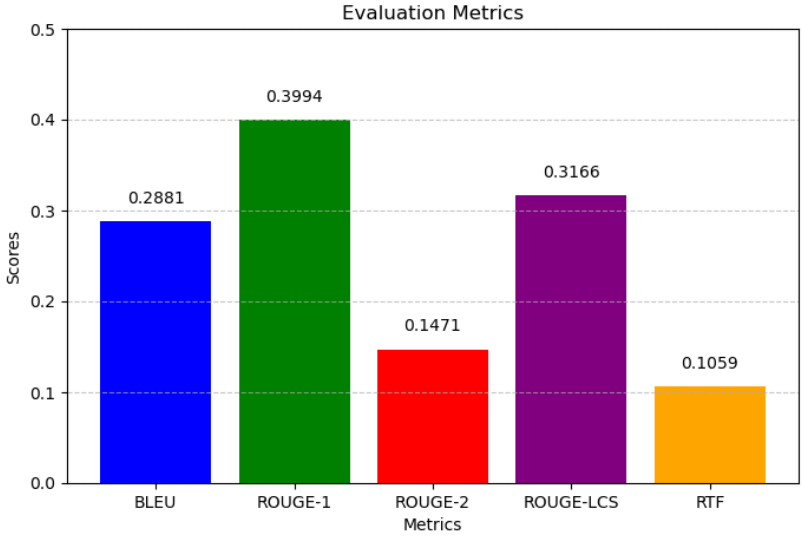

Figure 3: Task 2 - BLEU and ROUGE Scores for Single Headed Medusa

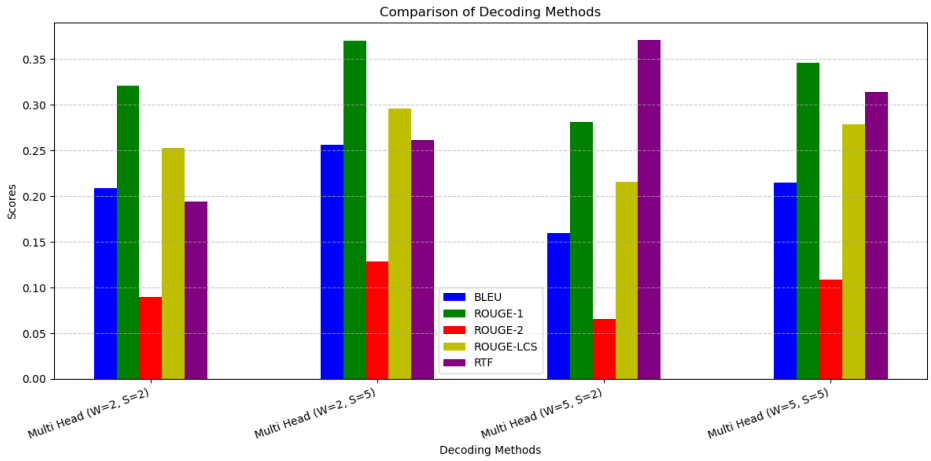

Figure 4: Task 2 - BLEU and ROUGE Scores for Multi Headed Medusa