Assignment 3 Report
CS-726: Advanced Machine Learning

Deeptanshu Malu Deevyanshu Malu Neel Rambhia

1 Task 0

Decoding Strategy	**BLEU**	**ROUGE-1**	**ROUGE-2**	**ROUGE-LCS**
Greedy	**0.3097**	**0.3538**	**0.1297**	**0.2704**
Random (temp=0.5)	0.2856	0.2929	0.1113	0.2387
Random (temp=0.9)	0.1996	0.1791	0.0550	0.1477
Top-k (k=5)	0.2388	0.2230	0.0607	0.1715
Top-k (k=10)	0.2427	0.2491	0.0799	0.2004
Nucleus Sampling (p=0.5)	0.2706	0.2554	0.0905	0.1973
Nucleus Sampling (p=0.9)	0.1957	0.1883	0.0469	0.1329

Table 1: Evaluation metrics for different decoding strategies.

Greedy decoding performs best across all metrics. On adding more randomness (higher temperature or larger sampling pools), the performance gets worse. Among sampling methods, Random with temperature=0.5 and Nucleus Sampling with p=0.5 work better than others. This tells us that some randomness can help, but too much hurts performance.

2 Task 1

The model used is `meta-llama/Llama-2-7b-chat-hf`.

We implemented constrained decoding using a trie data structure to limit the model's output to a predefined vocabulary. The algorithm selects the highest probability token that is valid according to the trie at each step.

We tested two trie traversal strategies: resetting the trie when reaching an end node (is_end) and resetting when a node has no children. The second strategy performed slightly better, as shown in the results table below.

Strategy	**BLEU**	**ROUGE-1**	**ROUGE-2**	**ROUGE-LCS**
Regular greedy	0.2781	0.3351	0.1230	0.2640
Reset when is_end	0.5111	0.5270	0.3189	**0.4694**
Reset when no children	**0.5116**	**0.5307**	**0.3205**	0.4675

Table 2: Evaluation metrics for constrained decoding strategies.

Algorithm 1 Constrained Decoding using Trie

```
1: Build trie from target vocabulary (word list)
2: Add EOS token to trie
3: $node \leftarrow trie.root$
4: for $i = 1$ to $max\_output\_length$ do
5:     Compute logits for the last token in the sequence
6:     Convert logits to probabilities using softmax
7:     Sort probabilities in descending order
8:     for each token $t$ in sorted order do
9:         if $t$ is a child of current $node$ then
10:            Select $t$ as the next token
11:            $node \leftarrow node.children[t]$
12:            if $node$ has no children then        ▷ Can be replaced with $node.is\_end$ condition
13:                $node \leftarrow trie.root$                              ▷ Reset trie state
14:            end if
15:            break
16:        end if
17:    end for
18:    Append selected token to the output sequence
19:    if selected token is EOS then
20:        break
21:    end if
22: end for
23: return generated sequence
```

3 Task 2

The model used is `FasterDecoding/medusa-v1.0-vicuna-7b-v1.5`. The results for different beam widths (W) and numbers of Medusa heads (S) are shown below:

W	S	BLEU	ROUGE-1	ROUGE-2	ROUGE-LCS	RTF
2	2	**0.1294**	**0.1174**	**0.0192**	**0.0987**	0.0207
5	2	0.0912	0.0970	0.0168	0.0782	0.0221
10	2	0.0510	0.0822	0.0122	0.0612	0.0252
2	5	0.0550	0.0794	0.0053	0.0721	0.0278
5	5	0.0261	0.0522	0.0042	0.0457	**0.0148**
10	5	0.0005	0.0169	0.0000	0.0149	0.0253

Table 3: Evaluation metrics for Medusa with different beam widths and head counts.

The best performance was achieved with the smallest beam width (W=2) and fewest heads (S=2). As we increased beam width or number of heads, performance declined across all metrics. This suggests that for this specific task, simpler configurations work better. The configuration with W=5 and S=5 achieved the best real-time factor (RTF), indicating faster decoding, but at a significant cost to generation quality.

Contributions

- **Deeptanshu Malu**:

  1.

- **Deevyanshu Malu**:

  1.

- **Neel Rambhia**:

  1.

Acknowledgements

- We have also used Copilot for faster coding and not for direct logic.