# Assignment 3 Report, CS 726: Spring 2024-25

Anupam Rawat
IIT Bombay
22b3982@iitb.ac.in

Aryan Gupta
IIT Bombay
22b2255@iitb.ac.in

Satush Parikh
IIT Bombay
21d070062@iitb.ac.in

March 25, 2025

## Contents

## 1 Introduction

The assignment requires the implementation of several techniques for sequence generation, given a Large Language Model (in this case - **Llama 2-7b-hf**), evaluated on the English-to-Hindi translation task with **IN22-Gen dataset** using **BLEU Score** and **ROUGE Score**.

### 1.1 Llama 2-7b-hf

The recommended model is a pretrained Large Language Model with 7B parameters.The model has been developed by Meta, which was trained between January 2023 and July 2023. Llama 2 is an auto-regressive language model that uses optimized transformer architecture. The model is intended for use in English, thus the task of machine translation from Hindi to English in the IN22-Gen Dataset is a challenging task.

### 1.2 Performance Metrics

1. **BLEU Scores (Bilingual Evaluation Understudy)** measures the precision of n-grams (unigrams, bigrams, trigrams, etc.) between the machine-generated translation and the reference translation. The metric focuses on precision and compares how many n-grams in the candidate translation match the reference translations.
   The score ranges between 0 to 1, higher score indicating a better translation quality. The metric tends to favor translations that use words that appear in the reference, but does not reward fluency, grammar, or meaning.
2. **ROUGE Score (Recall-Oriented Understudy for Gisting Evaluation)** is a set of metrics primarily used for evaluating summarization and text generation, but can also be applied to machine translation. It measures recall by comparing the overlap of n-grams between the reference and the candidate output.
   Similar to BLEU, ROUGE score also ranges between 0 to 1, higher score indicating a better recall / better overlap.
   (a) *ROUGE-1* measures unigram overlap (i.e. overlap of single words), between the reference and candidate translation.
   (b) *ROUGE-2* measures bigram overlap (i.e. overlap of a pair of consecutive words).
   (c) *ROUGE-LCS (Longest Common Sequence)* measures the longest common subsequence between the reference and candidate translation and focuses more on the sequence of words being in correct order, rather than plain overlaps. A higher ROUGE-LCS score suggests better preservation of the original meaning and structure in the translation.

### 1.3 IN22-Gen Dataset

The IN22-Gen is a dataset consisting of 1024 sentences sourced from Wikepedia and other online sources spanning various domains ranging from culture and tourism to education and government. The dataset has the translation of these 1024 sentences from English to 22 Indic Languages(Hindi is one amongst the 22 languages). The task for this assignment is to convert the sentences from Hindi to English.

## 2 Task 0: Introduction to LLM Decoding Techniques

Contrary to the widespread assumption, the model doesn't output a single word or a sentence. Firstly, the input text is encoded with a Tokenizer, then this input_token_ids, are fed as an input to the model. The model outputs logits, which can be converted to probabilities using softmax function. The task of the decoding techniques is to select one particular token out of the tokens given with their probabilities.
Below are the different widespread Decoding Techniques. These techniques are evaluated on two prominent metrics - **BLEU Score** and **ROUGE Score**, evaluated on the IN22 Dataset, for translation from Hindi

to English tasks. For each of the Decoding techniques, the outputs are generated iteratively until the End-Of-Sequence (EOS) token is reached.

## 2.1 Greedy Decoding

For the case of Greedy Decoding, we pick the token with the highest probability among the 32,000 options (vocabulary size of Llama-2-7b-hf is 32k). Mathematically, at the $t^{th}$ step, the next token can be obtained:

$$y_t = argmax_w P(w|y_{1:t-1}, x) \tag{1}$$

The token selected at $t^{th}$ is the token with the maximum probability, selected among the tokens generated when the input and the previously generated token are provided as input.
The greedy decoding technique, is not very far sighted and looks at just the next token. Thus, the greedy decoding technique produces moderate scores on the standard tasks.

## 2.2 Random Sampling with Temperature Scaling

Unlike, the Greedy Decoding method, where the most probable token was chosen, in this decoding method, we randomly sample from the probability distribution while adjusting its sharpness using a temperature parameter $\tau$. The probabilities are modified as follows and a token is randomly sampled from $P^{'}$.

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

At lower $\tau$ values, BLEU and ROUGE scores are relatively higher, indicating better coherence and fidelity since, the generated translations are less random, therefore favouring higher-probability words.
At medium $\tau$ values, a noticeable drop in BLEU and ROUGE can be observed due to the increasing randomness. As the $\tau$ continues to increase, we observe a significant decrease in all scores, especially ROUGE-2 and ROUGE-LCS, which measure phrase coherence, suggesting that the model is choosing words too randomly.
As the tau increases, we observe a trade-off between accuracy and diversity. A value of $\tau = 0.6$ seems to be best choice, balancing fluency and diversity.

## 2.3 Top-k Sampling

For the top-k decoding sampling technique, we choose the next token, based on the top-k most probable tokens. Firstly, we take the top-k tokens having the highest probabilities : $V_k = w_1, w_2, ..., w_k$, where $P(w_i) \geq P(w_{i+1})$ for $i < k$. Then the probabilities are normalized and a token is chosen randomly from $P^{'}$, until the generation of an EOS token:

$$P^{'}(w) = \begin{cases} \frac{P(w)}{\sum_{w' \in V_k} P(w')}, & \text{if w} \in V_k \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Now, it may be expected that having a larger k means, that the chances of choosing a token from the sample having a lesser probability increases and therefore the next token, might not provide the appropriate translation. But at the same time, having a larger k, also increases the diversity of the tokens, allowing you to have more meaningful representations. The value of $\mathbf{k = 6}$, finds the perfect match between diversity and meaningfulness, therefore giving the best scores.

## 2.4 Nucleus Sampling

Unlike previous methods, which relied on a fixed number of tokens, in this case, we dynamically choose the smallest set of tokens whose cumulative probability exceeds a threshold p, and then normalize the probabilites:

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

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

Lower p-values correspond to higher BLEU & ROUGE Scores, with **p = 0.5** achieving the best overall performance, indicating a balance between coherence and controlled diversity. Constraining the nucleus size ensures allows high-probability words to dominate while still allowing some variation.
As the p value increases to 0.7, the scores slightly decline, since more low-probability words are introduced and translations start diverging from reference texts. As the p value continues to increase, we see a sharp drop in the scores indicating that now along with diversity, a lot of randomness has also crept in, therefore impacting translation accuracy.
A p value of 0.5 is optimal as it ensures high-quality, fluent, and coherent translations. And a p value of 0.6-0.7 can be used when slightly more diverse translations are acceptable.

## 2.5 Results and Conclusion

The results highlight the trade-offs between determinism, diversity, and fluency in different decoding techniques for Hindi-to-English translation using LLaMA 2.

- Greedy decoding achieves the highest BLEU and ROUGE-1 but lacks diversity, often leading to repetitive outputs.
- Random sampling with temperature scaling ($\tau = 0.5$) performs well but declines at higher temperatures due to excessive randomness.
- Top-k sampling (k = 6) achieves a balance but struggles with coherence at higher k-values.
- Nucleus sampling (p = 0.5) offers a strong trade-off, maintaining coherence while allowing some diversity.

Overall, Greedy decoding is best for accuracy, while nucleus sampling (p = 0.5) provides the best balance between fluency and diversity.
These tasks were also run on the **Llama-3.1-8B** parameters, which produced a far better results. This is due to the fact that, this model is trained on a more diverse dataset, uses a better tokenization strategy, and has 1 Billion more parameters compared to Llama-2, therefore giving overall better results.

# 3 Task 1: Word Constrained Decoding

## 3.1 Introduction

Typically in sentences, once you have a starting portion, the range of possible next tokens decreases rapidly, given the sentence formation, structure, setting, and other factors. **Grammar Constrained Decoding** uses this property to get more accurate outputs from the LLM. In this exercise, we implement an alternate version, **Word-Constrained Decoding**, where we utilize a predefined set of words ("bag of words") to guide the output generation and therefore improve the performance of the LLM.

## 3.2 Methodology

Now, a word neccessarily can't be represented by a signle token, it might require a sequence of tokens for complete representation. Thus, to implement Word-Constrained Decoding, we employ a **Trie**-based approach to efficiently manage valid word sequences. The main steps of the algorithm are as follows:

1. **Building the Trie**: The bag of words is tokenized and inserted into a trie, where each node represents a token and also contains a flag at to indicate the completion of the word.
2. **Next Token** is chosen using the trie structure, to account for a valid word and via greedy decoding.
3. **Valid Token Masking**: At each step, only tokens corresponding to valid next token are considered.
4. **End-of-Word Handling**: Upon reaching the end of a word in the trie, the decoding process resets.

## 3.3 Implementation Details

### 3.3.1 TrieNode

The Trie is implemented using *TrieNode*. Each TrieNode consists of a dictionary which stores the the relationship of parent-child in a key-value pair manner. Each TrieNode, also consists of a flag indicating, whether it is end of the word.

### 3.3.2 TokenTrie

The *TokenTrie* class stores sequences of tokenized words to enforce constraints during decoding. It consists of the following key methods:

- **insert(token_sequence)**: Inserts a sequence of tokens into the trie and flags the last token.
- **get_next_tokens(node)**: Retrieves the next possible tokens from the given node in the trie.
- **is_complete_word(token_sequence)**: Checks if sequence of tokens forms a complete word.

### 3.3.3 ConstrainedTextGenerator

The class *ConstrainedTextGenerator* builds a Trie, and integrates it with language model to perform constrained output generation, giving controlled yet flexible results.

- **_build_trie_from_word_list()** takes the word_list and converts it into a Trie datastructure.
- **__call__()** implements the word-constrained decoding using trie, masking and greedy decoding technique.
    1. Initializes the TokenTrie and sets the current node to the root.
    2. Iteratively generates tokens up to max_output_len:
        - Concatenates previously generated tokens with input tokens.
        - Passes the sequence through the language model to obtain token logits.
        - Extracts allowed next tokens from the Trie.
        - Applies masking to restrict the output space.
        - Selects the most probable token (greedy decoding strategy).
        - Updates the Trie state to track word formation.
    3. Terminates generation upon reaching the end-of-sequence token (eos_token_id) or exceeding max_output_len.

## 3.4 Results and Conclusion

The results mentioned in Table-1, demonstrate the effectiveness of Word-Constrained Decoding in improving output alignment with the reference text by enforcing a strict vocabulary constraint. The method produces better results as compared to the standard greedy decoding technique employed in previous task.

| Task | BLEU | ROUGE-1 | ROUGE-2 | ROUGE-LCS |
|---|---|---|---|---|
| **LLM Decoding Techniques** | | | | |
| Greedy Decoding | **0.31** | **0.35** | **0.13** | **0.27** |
| Random Sampling with Temperature Scaling | | | | |
| $\tau = 0.5$ | **0.29** | **0.30** | **0.11** | **0.24** |
| $\tau = 0.6$ | 0.28 | 0.29 | 0.10 | 0.23 |
| $\tau = 0.7$ | 0.26 | 0.26 | 0.09 | 0.20 |
| $\tau = 0.8$ | 0.21 | 0.19 | 0.06 | 0.15 |
| $\tau = 0.9$ | 0.20 | 0.18 | 0.05 | 0.15 |
| Top-k sampling | | | | |
| k = 5 | 0.24 | 0.23 | 0.06 | 0.17 |
| k = 6 | **0.24** | **0.24** | **0.07** | **0.18** |
| k = 7 | 0.23 | 0.21 | 0.07 | 0.15 |
| k = 8 | 0.22 | 0.19 | 0.05 | 0.15 |
| k = 9 | 0.23 | 0.20 | 0.05 | 0.16 |
| k = 10 | 0.22 | 0.22 | 0.05 | 0.17 |
| Nucleus Sampling Decoding Techniques | | | | |
| p = 0.5 | **0.28** | **0.31** | **0.10** | **0.25** |
| p = 0.6 | 0.26 | 0.25 | 0.10 | 0.22 |
| p = 0.7 | 0.25 | 0.26 | 0.09 | 0.20 |
| p = 0.8 | 0.22 | 0.23 | 0.08 | 0.18 |
| p = 0.9 | 0.19 | 0.19 | 0.05 | 0.15 |
| **Word Constrained Decoding** | **0.55** | **0.59** | **0.33** | **0.49** |

Table 1: Performance Metrics for different decoding techniques for Llama-2-7b-hf

| Decoding Technique | BLEU | ROUGE-1 | ROUGE-2 | ROUGE-LCS |
|---|---|---|---|---|
| Greedy Decoding | **0.41** | **0.48** | **0.26** | **0.41** |
| Random Sampling with Temperature Scaling | | | | |
| $\tau = 0.5$ | **0.43** | 0.44 | 0.24 | 0.38 |
| $\tau = 0.6$ | 0.41 | **0.46** | **0.24** | **0.39** |
| $\tau = 0.7$ | 0.42 | 0.44 | 0.23 | 0.38 |
| $\tau = 0.8$ | 0.36 | 0.39 | 0.18 | 0.33 |
| $\tau = 0.9$ | 0.38 | 0.39 | 0.19 | 0.34 |
| Top-k sampling | | | | |
| k = 5 | **0.42** | 0.41 | 0.20 | 0.36 |
| k = 6 | 0.41 | **0.44** | **0.22** | **0.37** |
| k = 7 | 0.40 | 0.43 | 0.18 | 0.35 |
| k = 8 | 0.39 | 0.40 | 0.18 | 0.33 |
| k = 9 | 0.35 | 0.35 | 0.14 | 0.29 |
| k = 10 | 0.33 | 0.38 | 0.17 | 0.33 |
| Nucleus Sampling Decoding Techniques | | | | |
| p = 0.5 | 0.38 | 0.43 | 0.23 | 0.37 |
| p = 0.6 | 0.40 | 0.43 | 0.23 | 0.37 |
| p = 0.7 | **0.42** | **0.44** | **0.25** | **0.39** |
| p = 0.8 | 0.38 | 0.43 | 0.22 | 0.38 |
| p = 0.9 | 0.40 | 0.42 | 0.21 | 0.37 |

Table 2: Performance Metrics for different decoding techniques for Llama-3.1-8B

# 4 Task 2: Staring into Medusa's Heads

## 4.1 Introduction to Parallel Decoding

Traditional autoregressive language models generate text sequentially, predicting each token conditioned on all previously generated tokens. While effective, this approach suffers from high latency since each token requires a full forward pass through the model.

Medusa introduces a novel extbfspeculative parallel decoding paradigm that enables the model to predict multiple future tokens simultaneously through specialized parallel prediction heads. This approach significantly accelerates text generation while maintaining reasonable accuracy, making it particularly useful for applications requiring low-latency responses, such as real-time translation and conversational AI.

## 4.2 Medusa Architecture

The Medusa framework enhances standard transformer models by integrating two crucial components:

- **Multiple Prediction Heads**: Additional parallel output heads $(H_1, H_2, ..., H_K)$ that predict future tokens simultaneously.
- **Verification Module**: A mechanism to validate the predicted token sequences using the base model's probabilities.

For an input sequence $x_1, x_2, ..., x_t$, the base model computes the hidden states $h_t$. Each Medusa head $H_k$ then predicts the probability distribution of future tokens:

$$P^{(k)}(x_{t+1}, ..., x_{t+k} \mid x_{1:t}) = \prod_{i=1}^{k} \text{softmax}(W_i^{(k)} h_t) \tag{4}$$

where $W_i^{(k)}$ are learned projection matrices for position $i$ in head $k$. These projections allow each head to independently predict a fixed number of tokens ahead.

## 4.3 Medusa Decoding Algorithm

The complete Medusa decoding process follows an iterative approach, leveraging multiple prediction heads to enhance generation speed while ensuring accuracy via verification. The procedure is as follows:

```
Require: Input prompt $x_{1:n}$, maximum length $L$, beam width $B$, number of Medusa heads $K$
1:  Initialize candidate beam set $C \leftarrow \{x_{1:n}\}$
2:  while $|x| < L$ and no EOS token in all candidates do
3:      for each candidate sequence $c \in C$ do
4:          Compute base model hidden states $h_t$ for $c$
5:          Use $K$ Medusa heads to generate $M$ candidate continuations $S = \{s_1, ..., s_M\}$
6:          Compute scores for each candidate:
                $s(m) = \sum_{i=1}^{k} \log P_{\text{medusa}}^{(i)}(x_{t+i})$        (5)
7:      end for
8:      Select top-$B$ candidates by score
9:      Verify candidates using base model probabilities:
10:     for each candidate $s_m \in S$ do
11:         if $P_{\text{base}}(x_t) \geq \alpha \cdot P_{\text{medusa}}(x_t)$ for all tokens then
12:             Accept $s_m$ into updated beam $C$
13:         else
14:             Discard $s_m$
15:         end if
```

```
16:     end for
17: end while
18: return Best sequence from beam $C$
```

## 4.4 Key Components

### 4.4.1 Speculative Parallel Generation

Each Medusa head generates multiple token candidates simultaneously, significantly reducing generation time. For example, with $K = 4$ heads, we can predict 4 future tokens in parallel instead of processing them sequentially.

### 4.4.2 Tree-Based Verification Mechanism

Candidate sequences are structured into a prefix tree, where each node represents a token position. The verification phase follows a two-step process:

- Compute base model probabilities for each candidate token sequence.
- Accept tokens satisfying the condition:

$$P_{\text{base}}(x_t) \geq \alpha \cdot P_{\text{medusa}}(x_t) \tag{6}$$

  where $\alpha$ is a predefined acceptance threshold.

Tokens failing the acceptance criteria are discarded, ensuring high-quality outputs.

### 4.4.3 Beam Search Integration

The algorithm maintains multiple promising candidates simultaneously, refining selections through:

$$\text{Score}(c) = \sum_{i=1}^{t} \log P(x_i \mid x_{<i}) + \gamma \sum_{j=1}^{k} \log P_{\text{medusa}}(x_{t+j}) \tag{7}$$

where $\gamma$ controls the weight given to Medusa predictions, balancing efficiency and accuracy.

## 4.5 Implementation Details

Our implementation optimizes performance through:

- **Adaptive Head Selection**: Dynamically enables Medusa heads based on prediction confidence.
- **Memory-Efficient Parameter Sharing**: Medusa heads share base model parameters via linear adaptations.
- **Cache Optimization**: Hidden states are reused across prediction heads to minimize redundant computations.

## 4.6 Results Analysis

Table 3 presents an evaluation of Medusa decoding across different configurations:

- **Speed-Accuracy Tradeoff**: Medusa achieves a 3-5x speedup (RTF 0.03 vs 0.07) at the cost of a BLEU score reduction (0.12 vs 0.29).
- **Beam Width Effect**: Larger beam widths improve accuracy slightly but increase computation cost.
- **Optimal Head Count**: Increasing the number of Medusa heads improves parallelism but also raises the risk of cumulative errors.

The optimal configuration balances speed and accuracy, with beam width 2 and 2 Medusa heads achieving an RTF of 0.03 while retaining 41% of the base model's BLEU score.

### 4.7 Conclusion

Medusa decoding presents a promising technique for accelerating large language model inference by leveraging parallel token prediction. Despite minor accuracy trade-offs, its modular design allows for further refinements in:

- Improved head architectures to reduce prediction errors.
- More sophisticated verification mechanisms to enhance token acceptance.
- Adaptive parameter tuning for better speed-accuracy balance.

This technique is particularly beneficial for latency-sensitive applications, such as machine translation and dialogue systems, where fast response times outweigh minor reductions in output fidelity.

| Decoding Technique | BLEU | ROUGE-1 | ROUGE-2 | ROUGE-LCS | RTF |
|---|---|---|---|---|---|
| **Single Medusa Head** | 0.29 | 0.40 | 0.15 | 0.32 | 0.07 |
| **Multiple Medusa Heads** | | | | | |
| Beam-Width: 2, Medusa-Heads: 2 | 0.12 | 0.30 | 0.09 | 0.22 | 0.03 |
| Beam-Width: 2, Medusa-Heads: 5 | 0.12 | 0.30 | 0.09 | 0.22 | 0.03 |
| Beam-Width: 5, Medusa-Heads: 2 | 0.11 | 0.29 | 0.10 | 0.23 | 0.08 |
| Beam-Width: 5, Medusa-Heads: 5 | 0.11 | 0.29 | 0.10 | 0.23 | 0.07 |
| Beam-Width: 10, Medusa-Heads: 2 | 0.11 | 0.28 | 0.10 | 0.22 | 0.16 |
| Beam-Width: 10, Medusa-Heads: 5 | 0.11 | 0.28 | 0.10 | 0.22 | 0.17 |

Table 3: Performance Metrics for Medusa

## 5 AI Contribution

In the development of this project, **ChatGPT** & **Deepseek** was utilized for occasional debugging and assistance in the implementation of various functions. The LLMs contributed by providing initial drafts for these functions, which were then reviewed, debugged, and refined to align with the problem requirements.

## 6 Participant Contributions

The project involved multiple stages of development, debugging, and report preparation. The following contributions were made by the participants:

### 6.1 Aryan's Contributions

- Made the Report for the Assignment.
- Debugging and refining the implementations of :
  - Random Sampling with Temperature Scaling of Task 0 : Introduction to LLM Decoding Techniques.
  - Top-k Sampling of Task 0 : Introduction to LLM Decoding Techniques.

### 6.2 Anupam's Contributions

- Implemented and debugged the code for :
  - Task 0 : Introduction to LLM Decoding Techniques
  - Task 1 : Word Constraint Decoding

  - Multiple Head Decoding for Task 2 : Staring into Medusa's Heads.

- Made the Report for the project.

### 6.3 Satush's Contributions

- Debugging and refining the implementations of:
  - Implemented and debugged the code for initial part of multiple head decoding for Task 2 : Staring into Medusa's Heads.
  - Initial part of multiple head decoding for Task 2 : Staring into Medusa's Heads

The combination of AI-assisted function generation and participant debugging ensured that the project met its intended objectives.

## 7 References

- Medusa: Parallel Decoding for LLMs
- Attention is All You Need