CS726: Programming Assignment 3

Total Points: 50 + 10 (bonus)

October 11, 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_task0.py [to be changed]
  +- task1.py
  +- generate_task1_is.py [to be changed]
  +- task2.py
  +- generate_task2_smc.py [to be changed]
  +- task3.py
  +- generate_task3_tsmc.py [to be changed]
  +- api.py [to be changed]
  +- plot_histogram [to be implemented]
  +- eval.py
  +- util.py
  +- data
    +- test_prompts.jsonl
  +- 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 Dr. Shashi Tharoor and LLMs

Assume you are an LLM researcher on Earth-616, a timeline that houses a variant of Dr. Shashi Tharoor, endowed with the same prodigious mastery over the English language that puts a thesaurus to shame. This particular variant, however, harbors a deep disdain for the Llama [1] model for one reason: the model always insists on producing sentences that are far too simple and plain. To him, such linguistic modesty is an act of unforgivable dullness.

He has now turned to you for a remedy. Your mission, should you choose to accept it, is to modify the LLM's generation process so that its sentences become deliberately intricate, rich in vocabulary, and unnecessarily complex. That is, instead of generating *"Sampling is fun"*, the model must now produce *"The stochastic procurement of representative data subsets engenders a euphoric cognitive engagement of unparalleled delight"*. Let us now explore how such an outcome can be achieved.

Mathematically, let $P_{\text{llama}}(\boldsymbol{x} \mid x_0)$ denote the base LLaMA model given to you, and let $\boldsymbol{x} = x_1, x_2, \ldots, x_T$ represent the input token sequence, and x_0 is the prompt. For simplicity, we can ignore the prompt and rewrite the probability as

$$P_{\text{llama}}(\boldsymbol{x}) = \prod_{t=1}^{T} P_{\text{llama}}(x_t \mid x_{1:t-1}).$$

Now, let $P_{\text{tharoor}}(\boldsymbol{x})$ be the corresponding target distribution we want to achieve. This target distribution can be mathematically defined as:

$$\boxed{P_{\text{tharoor}}(\boldsymbol{x}) \propto P_{\text{llama}}(\boldsymbol{x})\, e^{\beta R(\boldsymbol{x})}}$$

where $R(\boldsymbol{x})$ is the reward function and $\beta > 0$ controls the strength of the bias towards the reward. Although the reward function can be anything, for the ease of the assignment, let us consider the reward function to be the cumulative trigram token rarities. That is, the reward function is defined as:

$$R(\boldsymbol{x}) = \begin{cases} \frac{1}{T} \sum_{t=3}^{T} r(x_{t-2}, x_{t-1}, x_t) & \text{if } T \geq 3 \\ 0 & \text{otherwise} \end{cases}$$

where $r(x_{t-2}, x_{t-1}, x_t)$ quantifies the rarity (or uncommonness) of the trigram (x_{t-2}, x_{t-1}, x_t) and we multiply by $\frac{1}{T}$ to ensure that the overall reward is length-normalized, preventing longer sequences from being unfairly favored. Formally, the per-token reward can be defined as:

$$r(x_{t-2}, x_{t-1}, x_t) = \begin{cases} -\log p_{\text{tri}}(x_t \mid x_{t-2}, x_{t-1}) & \text{if } x_t \in \mathcal{C} \text{ and } x_{t-1} \in \mathcal{C} \text{ and } x_{t-2} \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where p_{tri} denotes a trigram frequency model estimated from the corpus given to you, which we denote as $\mathcal{C}$, representing the collection of sentences used to compute empirical trigram counts and conditional probabilities. Specifically, p_{tri} can be calculated as:

$$p_{\text{tri}}(x_t \mid x_{t-2}, x_{t-1}) \;=\; \begin{cases} \dfrac{\text{count}(x_{t-2}, x_{t-1}, x_t)}{\text{count}(x_{t-2}, x_{t-1})}, & \text{if } (x_{t-2}, x_{t-1}, x_t) \in \mathcal{C} \\ 0 & \text{otherwise.} \end{cases}$$

1.1 Evaluation protocol and metrics

Sampling protocol. To evaluate the sampling methods defined below, you are given $A = 20$ test prompts. For every prompt, you must generate $B = 8$ samples. Thus, for each method, you will be generating a total of $N = 160$ predictions, using fixed random seeds where applicable to ensure reproducibility. Let $\mathcal{S} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots \boldsymbol{x}^{(N)}\}$ be the set of samples generated. You will be reporting the following three metrics:

1. **Expected reward.** Measures how successful a method is at producing rare n-grams:

$$\widehat{\mathbb{E}}[R] \;=\; \frac{1}{N} \sum_{i=1}^{N} R(\boldsymbol{x}^{(i)}).$$

If importance weights w_i are available, then the expected reward is to be calculated as:

$$\widehat{\mathbb{E}}[R] \;=\; \sum_{i=1}^{N} \tilde{w}^{(i)}\, R\big(\boldsymbol{x}^{(i)}\big), \qquad \tilde{w}^{(i)} = \frac{w^{(i)}}{\sum_{j=1}^{N} w^{(j)}}.$$

2. **Perplexity.** Measures how grammatically and semantically coherent the generated sequences are according to the base LLaMA model P_{llama}. For each sequence, compute its negative log-likelihood (NLL):

$$\text{NLL}(\boldsymbol{x}^{(i)}) = -\sum_{t=1}^{T} \log P_{\text{llama}}(x_t^{(i)} \mid x_{1:t-1}^{(i)}).$$

The empirical mean NLL and per-token perplexity are then given by:

$$\overline{\text{NLL}} = \frac{1}{N}\sum_{i=1}^{N} \text{NLL}(\boldsymbol{x}^{(i)}), \qquad \widehat{\mathbb{E}}[\text{Perplexity}] = \exp\Big(\frac{\overline{\text{NLL}}}{T}\Big).$$

If importance weights are available, use the weighted estimates:

$$\overline{\text{NLL}} \;=\; \sum_{i=1}^{N} \tilde{w}^{(i)}\, \text{NLL}(\boldsymbol{x}^{(i)}), \qquad \widehat{\mathbb{E}}[\text{Perplexity}] = \exp\Big(\frac{\overline{\text{NLL}}}{T}\Big).$$

3. **Entropy.** Measures how varied the generated samples are across the N generations. First, estimate the marginal token distribution:

$$\hat{p}(w) = \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} \mathbf{1}\{x_t^{(i)} = w\},$$

and compute its entropy:

$$\widehat{\mathbb{E}}[H] = -\sum_{w \in \mathcal{V}} \hat{p}(w) \log \hat{p}(w).$$

where $\mathcal{V}$ is the token vocabulary of Llama.

Note:

- All metrics are computed on the continuation tokens only (i.e., the prompt is excluded). We will refer to these metrics collectively as the "numbers".
- The algorithms described below present only a high-level overview. You are expected to handle any implementation-specific challenges that arise. For instance, for long token sequences, the probability $P_{\text{llama}}(\boldsymbol{x})$ may underflow to zero; hence, computations should be performed in log-space for numerical stability. Such implementation details are omitted for brevity.

1.2 Task 0: Primer on LLM Decoding Techniques [10 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-3 [1] when evaluated according to section 1.1.

(a) **Greedy Decoding:** At every step, you simply pick the token with the highest probability from P_{llama}. Formally, at the t^{th} step, you obtain the next token as follows:

$$x_t = \arg\max_{y \in \mathcal{V}} P_{\text{llama}}(y \mid x_{1:t-1}, x_0)$$

where $x_{1:t-1}$ denotes previously generated tokens and x_0 is the input prompt. This process is repeated iteratively until either the end-of-sentence (EOS) token is generated or $T_{\max}$ number of steps is reached. Report these numbers. [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:

$$P'_{\text{llama}}(y \mid x_{1:t-1}, x_0) = \frac{P_{\text{llama}}(y \mid x_{1:t-1}, x_0)^{1/\tau}}{\sum_{y' \in V} P_{\text{llama}}(y' \mid x_{1:t-1}, x_0)^{1/\tau}}$$

A token is then randomly sampled from P'_{llama}. Like before, keep repeating this process until the EOS token is generated or max number of steps is reached. Here, you must generate B samples for every prompt and report the numbers. Additionally, experiment with $\tau = 0.5$, $\tau = 0.9$ 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:

$$V_k = \{y_1, y_2, ..., y_k\}, \quad \text{where } P_{\text{llama}}(y_j \mid x_{1:t-1}, x_0) \geq P_{\text{llama}}(y_{j+1} \mid x_{1:t-1}, x_0) \text{ for } j < k$$

The probabilities within V_k are then normalized as follows:

$$P'_{\text{llama}}(y \mid x_{1:t-1}, x_0) = \begin{cases} \frac{P_{\text{llama}}(y|x_{1:t-1}, x_0)}{\sum_{y' \in V_k} P_{\text{llama}}(y'|x_{1:t-1}, x_0)} & \text{if } y \in V_k \\ 0 & \text{otherwise} \end{cases}$$

A token is then randomly sampled from P'_{llama}. As before, repeat the process and generate B samples for every prompt. Here, experiment with $k = 5$, $k = 10$ and report your findings. [4 pts]

The metrics you report for these decoding strategies will serve as baselines. Next, we move on to more sophisticated sequential sampling methods.

1.3 Task 1: Sequential Importance Sampling (SIS) [10 pts]

This task is designed to familiarize you with importance sampling and its inefficiencies. As before, you will first generate N samples using the Top-k sampling implemented earlier. Now, for each of these generated samples, calculate the corresponding importance weight as shown in algorithm 1. You are free to explore different values for β. Finally, to visualize the inefficiency of importance sampling, plot a histogram of the normalized importance weights. The histogram should have 10 bins corresponding to weight intervals of width 0.1 (i.e., $[0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0]$). Report your findings.

Algorithm 1 Sequential Importance Sampling

Input: Generated samples $\mathcal{S}$, reward function $R(\cdot)$

$w = [\,]$
for each $x \in \mathcal{S}$ **do**
    w.append$(e^{\beta R(x)})$
end for
$w[i] \leftarrow \frac{w[i]}{\sum_{j=1}^{|S|} w[j]} \quad \forall i \in \{1, \mid S \mid\}$ ▷ normalize
return w

1.4 Task 2: Sequential Monte Carlo Sampling (SMC) [15 pts]

In this task, you will implement the SMC algorithm discussed in class and compare its performance against the baseline and naive SIS methods. The overall generation procedure is outlined in algorithm 2.

For this task, use the following definition for the unnormalized intermediate target distribution:

$$\pi_t(x_{1:t}) = \exp\big(\beta R(x_{1:t})\big)$$

where $R(x_{1:t})$ denotes the cumulative reward up to time step t. As in Task 1, you should report the numbers and also analyse the weight distribution by plotting the histogram.

Algorithm 2 Sequential Monte Carlo

Input: prompt $x_0, P_{\text{llama}}, P_{\text{tharoor}}$, intermediates $\{\pi_t\}_{t=1}^{T_{\max}}$

for $t = 1, \ldots, T_{\max}$ **do**
  for $k = 1, \ldots, B$ **do**
    Sample $x_t^{(k)} \sim P_{\text{llama}}\left(* \mid x_{1:t-1}^{(k)}\right)$ ▷ Use top-k sampling with K=10
    Compute incremental weights

$$w_t^{(k)} \leftarrow \frac{\pi_t\left(x_{1:t}^{(k)}\right)}{\pi_{t-1}\left(x_{1:t-1}^{(k)}\right)\ P_{\text{llama}}\left(x_t^{(k)} \mid x_{1:t-1}^{(k)}\right)}$$

  end for
  if $t < T_{\max}$ **then**
    for $k = 1, \ldots, B$ **do**
      $w_t^{(k)} \sim \text{Cat}\left(\left\{\frac{w_t^{(i)}}{\sum_{j=1}^{K} w_t^{(j)}}\right\}_{i=1}^{K}\right)$
      Set $x_{1:t}^{(k)} \leftarrow x_{1:t}^{(w_t^{(k)})}$
    end for
  end if
end for
Return: $\{(x_{1:T}^{(k)}, w_t^{(k)})\}_{k=1}^{B}$

1.5 Task 3: Twisted Sequential Monte Carlo Sampling (TSMC) [15 pts]

In this task, you will implement the TSMC algorithm, which augments the standard SMC procedure with twist functions ϕ to introduce some sort of look-ahead bias. This modification helps prevent early collapse by guiding the sampling process toward high-reward regions. The overall generation procedure is outlined in algorithm 3.

For this task, use the following definition for the twist function:

$$\phi_t(x_{1:t}) = \begin{cases} exp(\beta(R(x_{1:t}) + \mathbb{E}_{x_{t+1}\sim p_{\text{tri}}(\cdot|x_{t-1},x_t)}\left[r(x_{t-1}, x_t, x_{t+1})\right])) & \text{if } t \geq 2 \\ exp(\beta\mathbb{E}_{x_{t+1}\sim p_{\text{bi}}(\cdot|x_t)}\left[r(x_t, x_{t+1})\right]) & \text{if } 1 \leq t < 2 \\ 0 & \text{otherwise} \end{cases}$$

where p_{bi} and $r(x_t, x_{t+1})$ are the bigram variant of the reward function. Use the same definition of the intermediate target distribution $\pi_t(x_{1:t})$ as in Task 2. Finally, note that these expectations can be easily computed using the corpus $\mathcal{C}$. As in Tasks 1 and 2, report the numbers and analyze the weight distribution by plotting a histogram of the normalized weights.

Algorithm 3 Sequential Monte Carlo

Input: prompt $x_0, P_{\text{llama}}, P_{\text{tharoor}}$, intermediates $\{\pi_t\}_{t=1}^{T_{\max}}$, twist functions $\{\phi_t\}_{t=1}^{T_{\max}}$

for $t = 1, \ldots, T_{\max}$ **do**
  for $k = 1, \ldots, B$ **do**
    Sample $x_t^{(k)} \sim P_{\text{llama}}\left(* \mid x_{1:t-1}^{(k)}\right)$ ▷ Use top-k sampling with K=10
    if $t < T_{\max}$ **then**
      $$w_t^{(k)} \leftarrow \frac{\phi_t\left(x_{1:t}^{(k)}\right)}{\phi_{t-1}\left(x_{1:t-1}^{(k)}\right)\ P_{\text{llama}}\left(x_t^{(k)} \mid x_{1:t-1}^{(k)}\right)}$$
    else
      $$w_t^{(k)} \leftarrow \frac{\pi_t\left(x_{1:t}^{(k)}\right)}{\phi_{t-1}\left(x_{1:t-1}^{(k)}\right)\ P_{\text{llama}}\left(x_t^{(k)} \mid x_{1:t-1}^{(k)}\right)}$$
    end if
  end for
  if $t < T_{\max}$ **then**
    for $k = 1, \ldots, B$ **do**
      $$w_t^{(k)} \sim \text{Cat}\left(\left\{\frac{w_t^{(i)}}{\sum_{j=1}^{K} w_t^{(j)}}\right\}_{i=1}^{K}\right)$$
      Set $x_{1:t}^{(k)} \leftarrow x_{1:t}^{(w_t^{(k)})}$
    end for
  end if
end for
Return: $\{(x_{1:T}^{(k)}, w_t^{(k)})\}_{k=1}^{B}$

1.6 [Bonus] Task 4: Free Reign on Sampling [10 pts]

This section is entirely optional and intended for those seeking extra brownie points (and eternal glory). Here, you are encouraged to explore more sophisticated ideas in sampling, be it a better reward function, alternative or learnt twist functions, or spinoffs of the SMC and TSMC algorithms themselves. You are only allowed to modify the components related to the *sampling process*; the evaluation setup must remain untouched, i.e., altering the prompts, random seeds, or the model itself will result in disqualification.

If you attempt this section, clearly describe your modifications in the report, justify your design choices, and present the corresponding evaluation metrics. The Top-5 teams, ranked according to some weighted combination of the evaluation metrics, will receive the full **brownie points**.

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
- Probabilistic Inference in Language Models via Twisted Sequential Monte Carlo

References

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