Understanding Sampling Hyperparameters in Large Language Models

Milton Leal
AI Research Engineer
WillowTree's Data and AI Research Team (DART)
milton.leal@willowtreeapps.com

July 2024

Abstract

This study investigates the impact of sampling hyperparameters on the output quality and diversity of text generated by GPT-4o, a leading Large Language Model (LLM) developed by OpenAI. We analyze four critical hyperparameters—temperature, top-p (nucleus sampling), presence penalty, and frequency penalty—to understand their roles in text generation. Through mathematical formulations, numerical examples, and empirical experiments, we demonstrate how fine-tuning these parameters balances creativity, coherence, and accuracy. Our results show that a temperature of 1.5 and top-p value of 1 increased output diversity by 1400%, while presence and frequency penalties set between 0.5 and 1.5 can significantly reduced token repetition. Interestingly, we found that very high or very low frequency penalty settings can lead to nonsensical outputs, highlighting the importance of careful parameter tuning. These insights provide valuable guidance for optimizing LLM performance in various applications, though results may vary with different prompts. Understanding these parameters is essential for AI researchers and engineers to fully leverage the potential of LLM technology while maintaining output quality and relevance. The code for the experiments is available at https://github.com/lealmilton/llm-hyperparameters-analysis.

1 Introduction

Large Language Models (LLMs) like GPT-4o [1] from OpenAI generate text by sampling from a probability distribution over possible next tokens. The quality and diversity of the generated text are influenced by four key hyperparameters: temperature, top-p, presence penalty, and frequency penalty. Understanding these parameters is important for helping the model to achieve desired outputs, such as enhancing coherence, ensuring diversity, and maintaining accuracy. This is particularly relevant in applications like LLM-backed chatbots, where controlling the generation of more or less varied responses can be crucial.

In this article, we explore the theoretical foundations and practical implications of these hyperparameters. We begin by defining each parameter and providing mathematical formulations to illustrate their effects on the sampling process. Following this, we present a series of experiments designed to measure the impact of varying these hyperparameters on model outputs when using the API provided by OpenAI. Specifically, we investigate:

- **Temperature**: Controls the randomness of the sampling process. Lower temperatures produce more deterministic outputs, while higher temperatures increase variability.
- **Top-p (Nucleus Sampling)**: Limits the sampling pool to the smallest subset of logits whose cumulative probability is at least p. Higher values allow more diverse token selection.
- **Presence Penalty**: Decreases the probability of tokens that have already appeared in the sampled text, encouraging the generation of new tokens without being too aggressive.
- **Frequency Penalty**: Reduces the probability of tokens based on their frequency in the generated text, avoiding repetition more aggressively than the presence penalty.

Our experiments include generating outputs for very simple prompts such as "Name a city" and "Generate a three-paragraph story about a mysterious forest", with varying settings for each hyperparameter. We analyze the results using heatmaps to visualize the text generation trend in terms of unique outputs and token counts under different configurations. The insights gained from these experiments may help AI researchers and engineers who aim to optimize LLM performance for applications.

The remainder of this article is structured as follows: Sections 2 through 5 provide an in-depth look at the hyperparameters. Section 6 present a set of experiments and results, followed by a discussion in Section 7 that concludes our findings.

2 Temperature

In the context of LLMs, logits are the raw, unnormalized scores produced by the neural network's final layer for each possible next token in a text sequence. These logits are transformed into probabilities through a softmax function, which converts them into a probability distribution over the potential next tokens [2].

The softmax function plays a critical role in normalizing these scores. Given a set of logits, the softmax function ensures that the output probabilities sum to 1, making them interpretable as probabilities.

Temperature (T) is a scaling factor applied to the logits before applying the softmax function. It controls the randomness of the sampling process, influencing how confidently the model makes its predictions.

Given logits $\text{logit}(x_i)$ for each possible next token x_i, the probability $P(x_i)$ of selecting token x_i is computed as:

$$P(x_i) = \frac{e^{\frac{\text{logit}(x_i)}{T}}}{\sum_j e^{\frac{\text{logit}(x_j)}{T}}} \tag{1}$$

- $\text{logit}(x_i)$: The raw score for token x_i produced by the model.
- T: The temperature parameter. It scales the logits before they are passed through the softmax function.
- e: The base of the natural logarithm, used in the exponentiation process to ensure positive probabilities.
- $\sum_j e^{\frac{\text{logit}(x_j)}{T}}$: The normalization factor, summing over all possible tokens x_j to ensure that the probabilities sum to 1.

In the OpenAI API, temperature varies from 0 to 2. While (T) cannot be 0 because it is used in the denominator, in practice, OpenAI sets it to a very small number that is not disclosed.

It is also important to note that in practice, OpenAI uses log probabilities (logprobs) in all cases to avoid overflow. It's worth noting that the OpenAI API provides access to these logprobs, allowing developers to gain insights into the model's token selection process. While we won't explore this feature in depth in this article, accessing logprobs via the API opens up possibilities for more advanced applications and analyses.

2.1 Numerical Examples

Consider the prompt "Chocolate is" and we want to sample the next token. We attribute fictional probabilities to two possible next tokens: "delicious" and "bitter".

For simplicity, let:

$$\text{logit}(\text{"delicious"}) = 2 \quad \text{and} \quad \text{logit}(\text{"bitter"}) = 1$$

2.1.1 Low Temperature ($T = 0.00001$)

Given a very low temperature, we have:

$$P(\text{"delicious"}) = \frac{e^{\frac{2}{0.00001}}}{e^{\frac{2}{0.00001}} + e^{\frac{1}{0.00001}}} \approx \frac{\infty}{\infty + 0} \approx 1$$

$$P(\text{"bitter"}) = \frac{e^{\frac{1}{0.00001}}}{e^{\frac{2}{0.00001}} + e^{\frac{1}{0.00001}}} \approx \frac{0}{\infty} \approx 0$$

Here, $P(\text{"delicious"})$ is extremely high, showing almost certainty. If we generate the next token 100 times with this temperature, "Chocolate is delicious" will be generated 100 times.

This demonstrates how lower temperatures make the model's output distribution sharper and more confident. The model tends to pick the highest probability token more frequently, resulting in more deterministic outputs. This helps increase accuracy. For example, if we are sampling the next token for the prompt "The capital of France is", we want the model to output "Paris" as often as possible. Setting the temperature to 0 will increase accuracy in this case.

2.1.2 High Temperature ($T = 1$)

Given a higher temperature, we have:

$$P(\text{"delicious"}) = \frac{e^{2/1}}{e^{2/1} + e^{1/1}} \approx 0.73$$

$$P(\text{"bitter"}) = \frac{e^{1/1}}{e^{2/1} + e^{1/1}} \approx 0.27$$

Here, $P(\text{"delicious"})$ is 0.73 and $P(\text{"bitter"})$ is 0.27, showing more diversity. If we generate the next token 100 times with this temperature, "Chocolate is delicious" will be generated approximately 73 times, and "Chocolate is bitter" will be generated approximately 27 times.

This illustrates how higher temperatures make the output distribution flatter, increasing the randomness of the predictions. The model's probability distribution over the next tokens becomes more uniform, allowing less probable tokens to be sampled more often, thus producing more diverse and creative outputs.

While temperatures above 1 can increase diversity, it's important to note that extremely high temperatures (such as $T = 2$) can lead to undesirable results because the model's outputs become too random and lose coherence, making it difficult to generate meaningful or relevant text.

3 Top-p (Nucleus Sampling)

Top-p sampling, also known as nucleus sampling, is a technique used in language models to limit the sampling pool to the smallest subset of tokens whose cumulative probability is at least p. This method helps balance between diversity and quality in text generation [3].

Given a token vocabulary V and their corresponding probabilities $P(x_i)$ for each token x_i, top-p sampling selects a subset S of V such that:

$$S = \min\{S' \subseteq V : \sum_{x \in S'} P(x) \geq p\} \tag{2}$$

The probabilities of the selected tokens are then renormalized:

$$P'(x_i) = \begin{cases} \frac{P(x_i)}{\sum_{y \in S} P(y)} & \text{if } x_i \in S \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

In the OpenAI API, top-p varies from 0 to 1.

3.1 Numerical Example

Consider the prompt "The animal in the story is a" with a vocabulary of five possible next tokens: "dog", "cat", "mouse", "bird", and "fish", with their associated probabilities:

$$
\begin{aligned}
P(\text{"dog"}) &= 0.3 \\
P(\text{"cat"}) &= 0.25 \\
P(\text{"mouse"}) &= 0.2 \\
P(\text{"bird"}) &= 0.15 \\
P(\text{"fish"}) &= 0.1
\end{aligned}
$$

For $p = 0.7$, we select tokens until their cumulative probability exceeds 0.7:

$$
\begin{aligned}
P(\text{"dog"}) &= 0.3 \\
P(\text{"dog"}) + P(\text{"cat"}) &= 0.55 \\
P(\text{"dog"}) + P(\text{"cat"}) + P(\text{"mouse"}) &= 0.75 \quad \text{(exceeds 0.7)}
\end{aligned}
$$

So, $S = \{\text{"dog"}, \text{"cat"}, \text{"mouse"}\}$. We then renormalize the probabilities:

$$
\begin{aligned}
P'(\text{"dog"}) &= \frac{0.3}{0.75} \approx 0.4 \\
P'(\text{"cat"}) &= \frac{0.25}{0.75} \approx 0.33 \\
P'(\text{"mouse"}) &= \frac{0.2}{0.75} \approx 0.27 \\
P'(\text{"bird"}) &= P'(\text{"fish"}) = 0
\end{aligned}
$$

Top-p sampling allows for a dynamic cutoff based on the probability distribution of the next token, offering several advantages. It adapts by varying the number of tokens considered based on the model's confidence. This method maintains a balance between generating high-quality tokens and allowing for diversity, thus providing a quality-diversity balance. Additionally, by limiting the sampling pool, it can be more computationally efficient than considering all possible tokens. Top-p sampling also helps maintain text coherence by eliminating very unlikely tokens while still allowing for creativity.

Lower p values (e.g., 0.2) focus on the most probable tokens, leading to more coherent but less diverse text. Higher p values (e.g., 0.9) increase diversity but may include less relevant tokens. The choice of p depends on the application: lower p for accuracy and coherence (e.g., factual writing) and higher p for creativity (e.g., story generation).

4 Presence Penalty

Presence penalty is a hyperparameter that decreases the probability of tokens that have already appeared in the text, encouraging the generation of new tokens without being too aggressive. This technique helps to reduce repetition and increase diversity in the generated text [4]. A common misconception is that presence and frequency penalties as well (see next section) are applied to the entire context of the conversation, e.g. applied to all previous system, user and assistant messages. However, these penalties are only applied to the current message being sampled, not to the entire conversation history.

Given a set of tokens x_i with their corresponding probabilities $P(x_i)$, and a presence penalty factor α, the adjusted probability $P'(x_i)$ for each token is calculated as:

$$P'(x_i) = P(x_i) - \alpha \cdot I(x_i \in \text{context}) \tag{4}$$

where:

- α is the presence penalty factor
- $I(x_i \in \text{context})$ is an indicator function that equals 1 if x_i is in the context, and 0 otherwise

After applying the presence penalty, the probabilities are renormalized to ensure they sum to 1:

$$P''(x_i) = \frac{P'(x_i)}{\sum_j P'(x_j)} \tag{5}$$

In the OpenAI API, presence penalty varies from -2.0 to 2.0. Positive values penalize new tokens based on whether they appear in the text so far, increasing the model's likelihood to talk about new topics. Negative values increase the likelihood of the model repeating the same line verbatim.

4.1 Numerical Example

Consider a vocabulary of three tokens: "forest", "mysterious", and "adventure", with their associated probabilities:

$$\begin{aligned} P(\text{"forest"}) &= 0.4 \\ P(\text{"mysterious"}) &= 0.3 \\ P(\text{"adventure"}) &= 0.3 \end{aligned}$$

Assume "forest" has already appeared in the context, and we set a presence penalty $\alpha = 0.1$.

Applying the presence penalty:

$$\begin{aligned} P'(\text{"forest"}) &= 0.4 - 0.1 = 0.3 \\ P'(\text{"mysterious"}) &= 0.3 \\ P'(\text{"adventure"}) &= 0.3 \end{aligned}$$

Renormalizing the probabilities:

$$\begin{aligned} Z &= 0.3 + 0.3 + 0.3 = 0.9 \\ P''(\text{"forest"}) &= \frac{0.3}{0.9} \approx 0.333 \\ P''(\text{"mysterious"}) &= \frac{0.3}{0.9} \approx 0.333 \\ P''(\text{"adventure"}) &= \frac{0.3}{0.9} \approx 0.333 \end{aligned}$$

This example demonstrates how the presence penalty reduces the probability of the token "forest" that has already appeared, making it equally likely to be chosen as the other tokens in the next sampling step.

5 Frequency Penalty

Frequency penalty is a hyperparameter that reduces the probability of tokens based on their frequency in the generated text, avoiding repetition more aggressively than the presence penalty. This technique helps to ensure a more balanced usage of different words and phrases in the generated text [4].

Given a set of tokens x_i with their corresponding probabilities $P(x_i)$, and a frequency penalty factor β, the adjusted probability $P'(x_i)$ for each token is calculated as:

$$P'(x_i) = P(x_i) - \beta \cdot \text{count}(x_i) \tag{6}$$

where:

- β is the frequency penalty factor
- $\text{count}(x_i)$ is the number of times token x_i has appeared in the generated text so far

After applying the frequency penalty, the probabilities are renormalized to ensure they sum to 1:

$$P''(x_i) = \frac{P'(x_i)}{\sum_j P'(x_j)} \tag{7}$$

In the OpenAI API, frequency penalty varies from -2.0 to 2.0. Positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim. Negative values increase the likelihood of the model repeating the same line verbatim.

5.1 Numerical Example

Consider a vocabulary of three tokens: "tree", "shadow", and "whisper", with their associated probabilities and frequencies in the generated text so far. We will show the changes in probabilities as the token "tree" appears once, twice, and three times.

Assume the initial probabilities are:

$$\begin{aligned} P(\text{"tree"}) &= 0.4 \\ P(\text{"shadow"}) &= 0.3 \\ P(\text{"whisper"}) &= 0.3 \end{aligned}$$

and we set a frequency penalty $\beta = 0.1$.

1. After "tree" has appeared once:

$$\begin{aligned} P'(\text{"tree"}) &= 0.4 - 0.1 \cdot 1 = 0.3 \\ P'(\text{"shadow"}) &= 0.3 \\ P'(\text{"whisper"}) &= 0.3 \end{aligned}$$

Renormalizing the probabilities:

$$\begin{aligned} Z &= 0.3 + 0.3 + 0.3 = 0.9 \\ P''(\text{"tree"}) &= \frac{0.3}{0.9} \approx 0.33 \\ P''(\text{"shadow"}) &= \frac{0.3}{0.9} \approx 0.33 \\ P''(\text{"whisper"}) &= \frac{0.3}{0.9} \approx 0.33 \end{aligned}$$

2. After "tree" has appeared twice:

$$\begin{aligned} P'(\text{"tree"}) &= 0.4 - 0.1 \cdot 2 = 0.2 \\ P'(\text{"shadow"}) &= 0.3 \\ P'(\text{"whisper"}) &= 0.3 \end{aligned}$$

Renormalizing the probabilities:

$$\begin{aligned} Z &= 0.2 + 0.3 + 0.3 = 0.8 \\ P''(\text{"tree"}) &= \frac{0.2}{0.8} = 0.25 \\ P''(\text{"shadow"}) &= \frac{0.3}{0.8} = 0.375 \\ P''(\text{"whisper"}) &= \frac{0.3}{0.8} = 0.375 \end{aligned}$$

3. After "tree" has appeared three times:

$$\begin{aligned} P'(\text{"tree"}) &= 0.4 - 0.1 \cdot 3 = 0.1 \\ P'(\text{"shadow"}) &= 0.3 \\ P'(\text{"whisper"}) &= 0.3 \end{aligned}$$

Renormalizing the probabilities:

$$\begin{aligned} Z &= 0.1 + 0.3 + 0.3 = 0.7 \\ P''(\text{"tree"}) &= \frac{0.1}{0.7} \approx 0.14 \\ P''(\text{"shadow"}) &= \frac{0.3}{0.7} \approx 0.43 \\ P''(\text{"whisper"}) &= \frac{0.3}{0.7} \approx 0.43 \end{aligned}$$

This example demonstrates how the frequency penalty reduces the probability of tokens that have appeared more frequently, making less frequent tokens more likely to be chosen in the next sampling step.

6 Experiments

In this section, we present a series of experiments designed to measure the impact of varying hyperparameters on model outputs. The experiments utilize asynchronous API calls and reproduce each setup multiple times to ensure reliability.

6.1 Experiment 1: Temperature and Top-p

This experiment analyzes the effect of temperature and top-p on the diversity of GPT-4o outputs. Higher temperatures and top-p values are expected to increase the number of unique responses.

6.1.1 Setup

- **Prompt**: "Name a city. Return only the name of the city."
- **Hyperparameters**:
  - Temperatures: [0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2]
  - Top-p values: [0, 0.25, 0.5, 0.75, 1]
  - Number of runs: 100

6.1.2 Results

The heatmap in Figure 1 illustrates the number of unique outputs generated for each combination of temperature and top-p values.

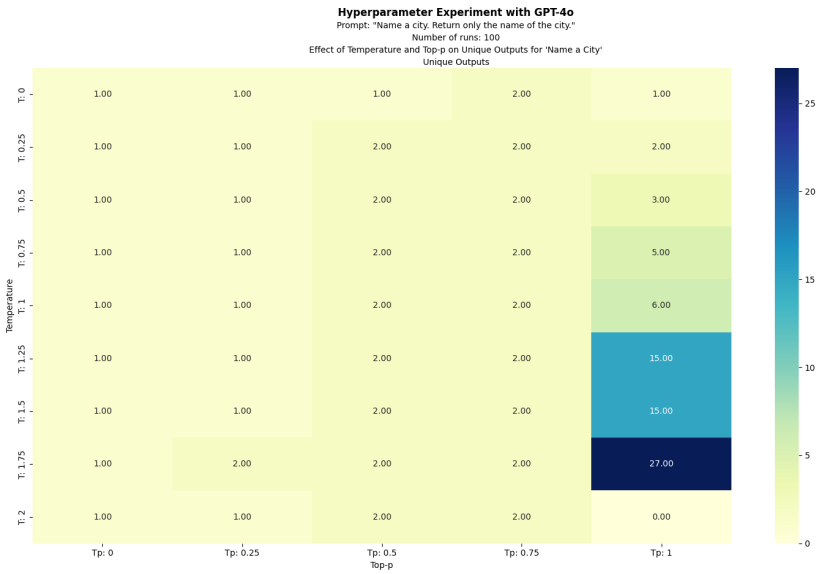

Figure 1: Effect of Temperature and Top-p on Unique Outputs for the Prompt "Name a City"

Here are the key observations from the experiment:

- **Low Temperature (T = 0)**: For almost all values of top-p, the model consistently generated a single unique output, with the exception being top-p = 0.75, indicating highly deterministic behavior, as expected with

a low temperature setting. This example also highlights that setting the temperature to zero does not mean absolute determinism; some variability may still be found.

- **Moderate Temperature (T = 0.5, 1)**: At these temperature settings, the diversity of outputs began to increase slightly with higher top-p values. The model produced 2 to 6 unique outputs, reflecting a balance between randomness and determinism.
- **High Temperature (T = 1.5)**: At higher temperatures, especially with a top-p value of 1, the model generated a significantly higher number of unique outputs (up to 15). This demonstrates the increased variability and creativity in the responses due to the higher temperature.
- **Extremely High Temperature (T = 1.75, 2)**: Although it seems that at a temperature of 1.75 the model generated the most unique examples, it is important to note that some of these outputs were considered unique because they were accompanied by a period (e.g., "Paris.") instead of just the name of the city. These examples, along with those when the temperature was set to 2 (when the API showed an error), illustrate the need for caution when using very high temperatures.
- **Top-p Influence**: Lower top-p values (0, 0.25, 0.5) resulted in fewer unique outputs across all temperature settings, indicating a more restricted sampling pool. Higher top-p values (0.75, 1) allowed for greater diversity, particularly at higher temperatures.

We calculated the percentage increase in unique outputs for specific hyperparameter setups compared to a base case (Temperature = 0, Top-p = 0). Here are the results:

- Temp = 0, Top-p = 1: 0.0% increase
- Temp = 0.5, Top-p = 1: 200.0% increase
- Temp = 1, Top-p = 1: 500.0% increase
- Temp = 1.5, Top-p = 1: 1400.0% increase

The results confirm that temperature and top-p are crucial for controlling the diversity of GPT-4o's outputs. Higher temperatures and top-p values increase randomness and expand the sampling pool, leading to more varied and creative responses. Lower values produce more predictable and deterministic outputs, suitable for applications requiring high coherence and accuracy.

6.2 Experiment 2: Presence Penalty

To investigate the impact of presence penalty on text diversity, we conducted multiple runs of the experiment varying presence penalty values for fixed combinations of temperature and top-p.

6.2.1 Setup

- **Prompt**: "Generate a three-paragraph story about a mysterious forest. Just include the story."
- **Hyperparameters**:
  - Temperatures and Top-p combinations: [(0,0), (0.25, 0.25), (0.5, 0.5), (0.75, 0.75), (1, 1)]
  - Presence penalties: [-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2]
  - Number of runs per combination: 10

6.2.2 Results

The heatmap in Figure 2 illustrates the average number of unique tokens generated for each presence penalty value. Here are the key observations from the experiment:

- **Negative Presence Penalty**: At presence penalty values of -2 and -1.5, the model generated fewer unique outputs, indicating highly repetitive behavior, as expected with negative presence penalty settings.
- **Neutral Presence Penalty (0)**: At a presence penalty of 0, the model produced a moderate number of unique outputs, reflecting the base diversity of the model without any penalty applied.
- **Positive Presence Penalty**: As the presence penalty increased from 0.5 to 2, the number of unique outputs increased significantly. For instance, at a presence penalty of 2, we observed a high number of unique tokens in the generated stories, demonstrating the increased diversity in the responses due to the higher presence penalty.

We calculated the percentage increase in unique tokens for specific hyperparameter setups compared to a base case (Temperature = 0, Top-p = 0, Presence Penalty = 0). Here are the results:

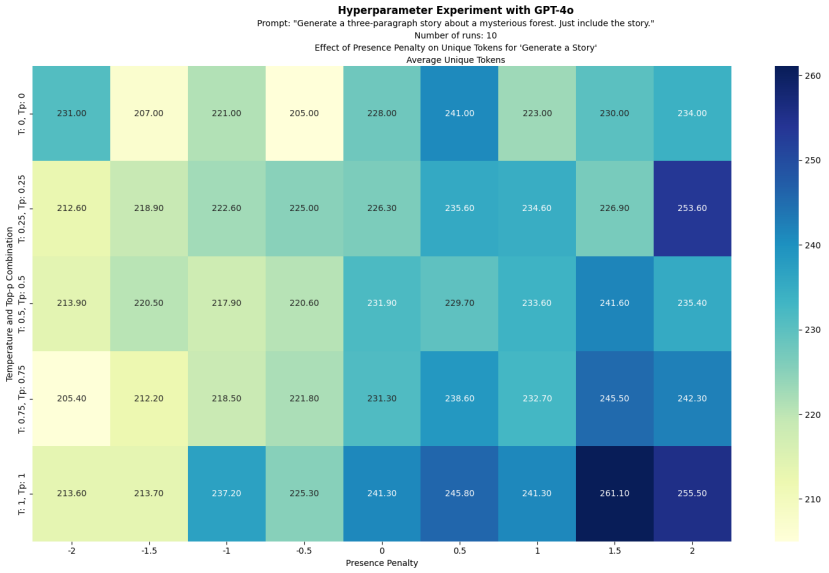

Figure 2: Effect of Presence Penalty on Unique Outputs for the Prompt "Generate a three-paragraph story about a mysterious forest. Just include the story."

- Temp = 0.25, Top-p = 0.25, Presence Penalty = 0.5: 3.33% increase
- Temp = 0.5, Top-p = 0.5, Presence Penalty = 0.5: 0.75% increase
- Temp = 0.75, Top-p = 0.75, Presence Penalty = 1.5: 7.68% increase
- Temp = 1, Top-p = 1, Presence Penalty = 1.5: 14.51% increase

The results confirm that presence penalty is key for controlling the diversity of GPT-4o's outputs. Higher positive presence penalty values increase the likelihood of generating new tokens, leading to more varied and diverse responses. Lower and negative values produce more repetitive and deterministic outputs.

6.3 Experiment 3: Frequency Penalty

To analyze the impact of frequency penalty on text diversity, we conducted multiple runs of the experiment varying frequency penalty values for fixed combinations of temperature and top-p.

6.3.1 Setup

- **Prompt**: "Generate a three-paragraph story about a mysterious forest. Just include the story."
- **Hyperparameters**:
  - Temperature and Top-p combinations: [(0,0), (0.25, 0.25), (0.5, 0.5), (0.75, 0.75), (1, 1)]
  - Frequency penalties: [-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2]
  - Number of runs per combination: 10

6.3.2 Results

The heatmap in Figure 3 illustrates the average number of unique tokens generated for each frequency penalty value.

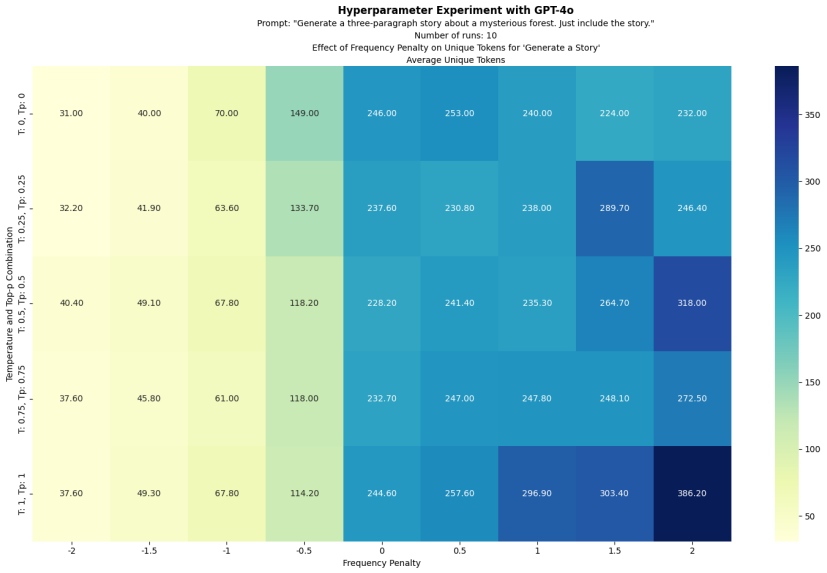

Figure 3: Effect of Frequency Penalty on Unique Outputs for the Prompt "Generate a three-paragraph story about a mysterious forest. Just include the story."

Here are the key observations from the experiment:

- **Negative Frequency Penalty**: At frequency penalty values of -2 and -1.5, the model generated fewer unique outputs, indicating highly repetitive behavior, as expected with negative frequency penalty settings. It's important to note that for these setups, the model generated nonsensical output. With such low frequency penalties, the model starts penalizing even white spaces.
- **Neutral Frequency Penalty (0)**: At a frequency penalty of 0, the model produced a moderate number of unique outputs, reflecting the base diversity of the model without any penalty applied.
- **Positive Frequency Penalty**: As the frequency penalty increased from 0.5 to 2, the number of unique outputs increased significantly. For instance, at a frequency penalty of 2, we observed a high number of unique tokens in the generated stories, demonstrating the increased diversity in the responses due to the higher frequency penalty. However, the text lost a lot of coherence when we used extremely high frequency penalty setups.

We calculated the percentage increase in unique tokens for specific hyperparameter setups compared to a base case (Temperature = 0, Top-p = 0, Frequency Penalty = 0). Here are the results:

- Temp = 0.25, Top-p = 0.25, Frequency Penalty = 0.5: 6.17
- Temp = 0.5, Top-p = 0.5, Frequency Penalty = 1: 4.34
- Temp = 0.75, Top-p = 0.75, Frequency Penalty = 1.5: 0.73
- Temp = 1, Top-p = 1, Frequency Penalty = 1: 20.69

The results show somewhat unexpected behavior for some values of frequency penalty and certain combinations of temperature and top-p. For positive frequency penalty, we expected to see an increase in the unique token generation, but we saw a flat trend for a lot of values, except when temperature and top-p were set to 1 or frequency penalty was set to higher (1.5 or 2) values.

At a frequency penalty of 2, we observed a significant increase in output diversity, indicating effective reduction of token repetition. However, very high or very low frequency penalty settings can produce nonsensical outputs. For example, settings like Temp: 0, Top-p: 0, Frequency Penalty: -2 resulted in repetitive and incoherent text generation. The same was true when we set Frequency Penalty: 2.

Based on our experiments, we recommend using frequency penalty values between 0.5 and 1.5 for generating diverse yet coherent outputs and also considering setting temperature and top-p to 1. Values outside this range may be experimented with caution, considering the trade-offs between diversity and coherence.

7 Conclusion

This work has provided an analysis of the effects of four key sampling hyperparameters—temperature, top-p, presence penalty, and frequency penalty—on the output diversity and coherence of text generated by GPT-4o. Through theoretical discussions and systematic experiments, we have demonstrated how these hyperparameters can be adjusted to influence the text generation process significantly.

Our findings reveal that higher temperature and top-p values result in more diverse outputs, with a temperature of 1.5 and top-p of 1 increasing unique outputs by 1400% compared to the base case. This makes these settings suitable for applications that require creative and varied text. Conversely, lower values lead to more deterministic outputs, which are preferable for tasks requiring high coherence and accuracy.

The presence and frequency penalties further refine the output by penalizing repetitive tokens, thereby enhancing the uniqueness of generated text. We found that presence penalty values between 0.5 and 1.5 led to increased diversity without compromising coherence, with a 14.51% increase in unique tokens at optimal settings. Interestingly, our experiments with frequency penalty revealed unexpected behaviors, including decreased diversity at certain combinations of temperature and top-p. We recommend using frequency penalty values between 0.5 and 1.5 for generating diverse yet coherent outputs.

A particularly striking finding was that extreme frequency penalty settings (-2 or 2) resulted in nonsensical or highly incoherent text, underscoring the importance of careful parameter tuning. This highlights the delicate balance between encouraging diversity and maintaining coherence in LLM outputs.

The insights derived from this study offer practical guidelines for optimizing LLM performance across various applications. By understanding the interplay of these hyperparameters, AI researchers and engineers can tailor the behavior of LLMs to meet specific requirements, whether for generating engaging narratives, maintaining conversational relevance, or ensuring factual consistency.

Future research could explore the combined effects of these hyperparameters in more complex scenarios and prompts, potentially uncovering additional nuances in their interactions. Additionally, extending this analysis to other language models could provide a broader perspective on the generalizability of our findings, contributing to a more comprehensive understanding of LLM behavior across different architectures and training paradigms.

References

[1] OpenAI. *Hello GPT-4o*. Accessed: 2024-06-30. 2024. URL: https://openai.com/index/hello-gpt-4o/.

[2] D Rothman. *Transformers for Natural Language Processing: Build Innovative Deep Neural Network Architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and More*. Packt Publishing, 2021. ISBN: 9781800565791. URL: https://books.google.com.br/books?id=Ua03zgEACAAJ.

[3] Haw-Shiuan Chang et al. *REAL Sampling: Boosting Factuality and Diversity of Open-Ended Generation via Asymptotic Entropy*. 2024. arXiv: 2406.07735 [cs.CL]. URL: https://arxiv.org/abs/2406.07735.

[4] Gonzalo Martínez et al. *Beware of Words: Evaluating the Lexical Richness of Conversational Large Language Models*. 2024. arXiv: 2402.15518 [cs.CL]. URL: https://arxiv.org/abs/2402.15518.