CSAI 422: Assignment 1 — Exploring Sampling Parameters in LLMs

Zeinab Montasser

202301128

GitHub repo: https://github.com/zeinabmontasser/Data-Mining-Assignment-1.git

Table of Contents

1. Introduction

This report documents an experiment conducted as part of the data mining assignment that explored how sampling parameters, specifically temperature, affect the outputs of large language models (LLMs). The experiment had two main parts: first, implementing the core sampling algorithms from scratch using NumPy to build a solid mathematical understanding of what these parameters actually do under the hood, and second, using the Groq API to generate real model responses across different temperature settings and observe the practical effects.

Temperature is one of the most fundamental parameters in LLM inference. It controls how deterministic or random the model's token sampling process is. At a low temperature, the model becomes very confident and predictable, almost always picking the highest-probability token. At a high temperature, the distribution flattens out and the model starts exploring less likely options, which tends to produce more varied and sometimes more creative output. Understanding this parameter, not just conceptually but mathematically is the whole point of this assignment.

Three apple-themed prompts were used across four temperature settings (0.0, 0.3, 0.7, and 1.0), with five responses generated per prompt per temperature, for a total of 60 API responses. The goal was to directly observe how temperature changes the character of the model's output.

2. Methodology

The experiment used the llama-3.1-8b-instant model via the Groq API, accessed through an OpenAI-compatible client. The implementation followed two parallel tracks: building the sampling functions from first principles in Python, and running an API-based experiment to observe the real effects on generated text.

For the API experiment, three prompts were designed, all centered around apples, which kept the subject matter consistent while testing different types of tasks:

Prompt 1: "Describe a world where apples are used as currency. What would the economy look like?"

Prompt 2: "Write a short story about a golden apple that grants infinite wealth to its owner."

Prompt 3: "Create a business plan for a luxury apple orchard that caters to billionaires."

These prompts were chosen intentionally because they vary in task type. The first is analytical and world-building, the second is purely creative and narrative, and the third is structured and practical. Keeping all prompts in the same thematic domain meant any differences in output were more clearly attributable to temperature than to prompt complexity.

Each prompt was run at temperatures of 0.0, 0.3, 0.7, and 1.0, generating five responses per combination, for a total of 60 API calls. All results were saved as timestamped JSON files using the save_results() function. For the Part 2 implementations, a simulated 10-token vocabulary with a fixed logit array was used to test and visualize each sampling method.

For the sampling implementations, the following functions were built and tested: softmax_with_temperature, top_k_sampling (naive), top_k_sampling_efficient (using argpartition), top_p_sampling, logit_bias_sampling, and a combined sampler that applies all parameters in the correct pipeline order. Both the temperature_effect.png and sampling_comparison.png visualizations were generated to illustrate the behavior of each method.

3. Results

The most striking result from the API experiment occurred at temperature 0.0. Across all five runs for a given prompt, the model returned identical responses, word for word. For example, all five responses to the apple economy prompt at temperature 0.0 produced the exact same structured breakdown with the same ten numbered points, identical headers, and identical phrasing. This behavior aligns with theoretical expectations: at temperature 0, the model deterministically selects the highest-probability token at each step, eliminating stochastic sampling and resulting in zero variation.

The mathematical reasoning behind this behavior lies in how temperature scales the logits before the softmax function:

$$P_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

When $T \to 0$, the logits are divided by an increasingly small number. This dramatically magnifies the relative differences between them. For example, if logits 2 and 4 are divided by 0.0001, they become 20,000 and 40,000, expanding their difference from 2 to 20,000. After exponentiation in the softmax function, the largest logit dominates the denominator, causing its probability to approach 1 while all others approach 0. In the limit, the softmax function behaves like an argmax operator, producing a degenerate (one-hot) distribution. As a result, token selection becomes fully deterministic and entropy collapses to zero.

At temperature 0.3, slight variation began to emerge, but responses were still largely consistent in structure and content. The model would reliably produce a structured, well-organized answer with similar key themes, currency grading, spoilage challenges, seasonal fluctuations, but occasionally varied how it introduced sections or ordered its points. The core argument remained stable across runs.

At temperature 0.7, the variation became more noticeable. Different responses would emphasize different aspects of the scenario, introduce unique sub-concepts (for instance, one response at this temperature introduced the idea of an 'Apple Futures' market in creative detail, while another focused on social stratification around apple ownership), and the vocabulary and sentence structure started to diverge more clearly. Responses were still coherent and on-topic, but had more individual character.

At temperature 1.0, the diversity was at its peak. Responses showed the widest range of ideas, vocabulary, and structure. One response to the economy prompt invented a currency called 'Appli' managed by a 'Central Apple Authority (CAA)', which was a completely different framing than anything that appeared at lower temperatures. Another response at T=1.0 used the phrase 'Apple Economy (ABE)' as a branded concept and explored the idea of 'Tree Index' as an economic indicator. These were creative framings that never surfaced at lower temperatures. Crucially though, the responses were still coherent and relevant, temperature 1.0 with llama-3.1-8b-instant did not produce incoherent or broken text, just more varied and creative output.

For the narrative prompt (golden apple story), the temperature effect was even more pronounced. At T=0.0, all five stories shared the same basic plot arc and similar character descriptions. At T=1.0, stories diverged significantly in tone, character motivation, and ending, some ended on cautionary notes about greed, others on triumphant notes. This confirmed that higher temperature genuinely increases creative diversity in narrative tasks.

4. Discussion

The patterns across temperatures mapped cleanly onto the theoretical predictions from Part 2. Low temperature produced high predictability and consistency, useful for tasks where a reliable, structured answer is needed every time. High temperature produced more creative exploration but also more variation, useful for tasks where diversity and originality are valued over repeatability.

One interesting observation was that temperature 0.3 felt like a kind of sweet spot for analytical tasks. The responses were more varied than T=0.0 but still tightly structured and reliably accurate. For a task like writing a business plan, T=0.3 or T=0.7 would probably be the right choice, enough structure to be useful, enough variation to not feel like a copy-paste every time.

For creative writing tasks like the golden apple story, T=0.7 or T=1.0 clearly produced more interesting results. The stories at higher temperatures had richer narrative details, more distinctive voices, and more unexpected plot directions. The tradeoff is consistency, if you need to run the same creative prompt multiple times and want different outputs (for instance, generating story variations for a user to choose from), high temperature is ideal.

Response length was also somewhat affected by temperature. At T=0.0, the model tended to produce very consistent response lengths. At T=1.0, response lengths varied more, with some responses being more concise and others more expansive. This makes sense, at higher temperatures the model can wander into different structural choices, including whether to elaborate on a point or move on.

One thing that was surprising: even at T=1.0, the model never produced anything incoherent or off-topic. The prompts were fairly clear and the model stayed within the relevant domain throughout. This suggests that for this particular model and prompt style, T=1.0 is still a safe and productive setting, the concern about 'hallucination' or 'rambling' at high temperature may depend more on the prompt design than temperature alone.

From a practical standpoint, the temperature recommendations that emerge from this experiment are roughly: use T=0.0 when you need perfectly reproducible outputs (automated pipelines, structured data extraction), use T=0.3-0.5 for analytical tasks where some variation is acceptable, use T=0.7-1.0 for creative writing and brainstorming tasks.

5. Sampling Parameter Analysis

Logits and the Need for Softmax

When an LLM processes a prompt, its final layer produces a vector of raw scores, one for each token in the vocabulary. These are called logits. They are unbounded real numbers and have no direct probabilistic meaning on their own: a logit of 3.1 does not mean a 31% chance. To convert them into a valid probability distribution (where all values are between 0 and 1 and sum to 1), we apply the softmax function. Softmax computes $\frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$, which maps the logits through an exponential and normalizes them. The result is a proper probability distribution that the sampling step can use to draw the next token.

Mathematical Effect of Temperature

Temperature works by dividing the logits before softmax is applied. The formula is softmax(logits / T). When T is small (e.g., 0.1), dividing by T amplifies the differences between logits. After exponentiation, the largest logit dominates by a huge margin, collapsing the distribution to near-deterministic. In the temperature_effect.png visualization, you can see that at T=0.1, the token 'on' (which had the highest logit at 3.1) captures almost all the probability mass, around 0.93, while everything else is essentially zero. At T=2.0, dividing by a large number shrinks the differences between logits, making the distribution much flatter and closer to uniform. The key insight is that temperature doesn't change which token has the highest probability, it changes how much more probable that token is compared to the others.

Top-k vs Top-p Sampling

Top-k sampling restricts the model to the k tokens with the highest logits, masking everything else to -inf before softmax. The advantage is simplicity and computational predictability, you always know exactly how many tokens are in play. The disadvantage is inflexibility: setting k=3 when the top-3 tokens already cover 99% of the probability mass is fine, but setting k=3 when the top-3 tokens only cover 40% of the mass (a flatter distribution) arbitrarily cuts out many reasonable options.

Top-p (nucleus) sampling addresses this by dynamically choosing the number of tokens based on the cumulative probability. You keep the smallest set of tokens whose total probability meets or exceeds p. This means when the distribution is sharp, you might only keep 1-2 tokens. When it's flat, you keep more. The tradeoff is that the number of tokens kept varies, which makes the behavior less predictable and slightly harder to reason about. In general, top-p is considered more principled for language generation tasks because it adapts to the shape of the distribution.

Logit Bias: A Concrete Use Case

Logit bias lets you add a constant value to specific token logits before softmax, effectively boosting or suppressing particular tokens regardless of what the model would normally predict. A practical real-world use case: a customer service chatbot where the company wants to prevent the model from ever suggesting that users 'cancel' their subscription. By applying a large negative logit bias (e.g., -100) to the token IDs corresponding to 'cancel', 'cancellation', 'terminate', and similar terms, you can effectively ban those tokens from ever appearing in the output, without needing to post-process or filter the text. This is cleaner and more reliable than trying to catch the word after generation.

Connection Between Part 2 Implementations and API Parameters

The temperature, top_p, and logit_bias parameters in the Groq/OpenAI API are exact equivalents of the functions implemented in Part 2. The API abstracts away the logit array and vocabulary, we never see the raw logit vector, but inside the model, the exact same math runs. Calling the API with temperature=0.7 is mathematically identical to calling softmax_with_temperature(logits, 0.7) on the model's internal logit array. This connection makes the Part 2 implementations genuinely useful for understanding what happens when you change API parameters: they aren't just academic exercises, they're the actual computations being performed.

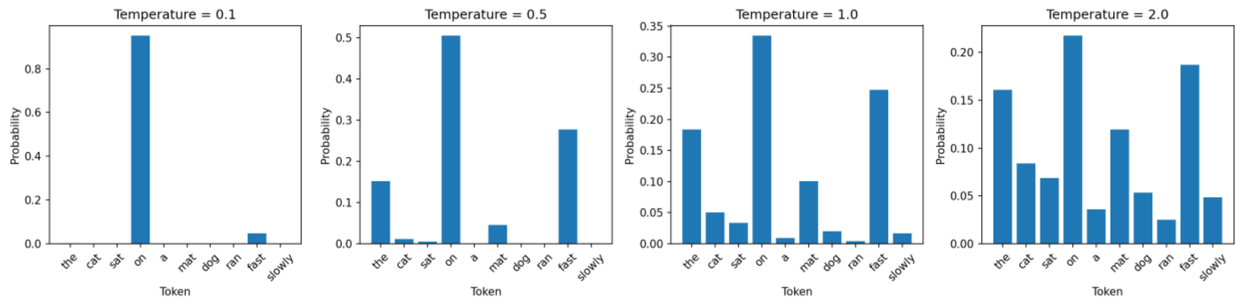

Fig 1: 4 panel bar chart comparing probability distribution with different temperature.

Efficient Top-k: Why O(V) vs O(V log V) Matters

The naive top-k implementation uses np.argsort to sort all V logits, which runs in O(V log V) time. The efficient implementation uses np.argpartition, which only guarantees that the top-k elements are in the last k positions of the array, it doesn't sort the full array, running in O(V) time. For a vocabulary of V = 1,000,000 tokens (which is not unrealistic for modern models), the difference is significant. At V = 1,000,000: O(V log V) ≈ 20,000,000 operations; O(V) ≈ 1,000,000 operations, a roughly 20x difference. Since token generation is sequential, every token requires one forward pass and one sampling step, and modern models generate tokens at rates of tens to hundreds per second, this efficiency gain compounds over the length of a response. For production-scale systems generating millions of tokens per day, the O(V) approach is not optional, it's necessary.

6. EXTRA CREDIT

The response length distribution demonstrates a clear relationship between temperature and structural variability. At temperature 0.0, responses exhibit the highest median length and the smallest variance, reflecting deterministic decoding behavior. As temperature increases, the median length gradually decreases while variance increases. At temperature 1.0, responses show the widest spread and lower median length, indicating more diverse continuation paths. This aligns with the theoretical interpretation of temperature as a scaling factor on logits: higher temperature flattens the probability distribution, increasing entropy and allowing more diverse token selection.

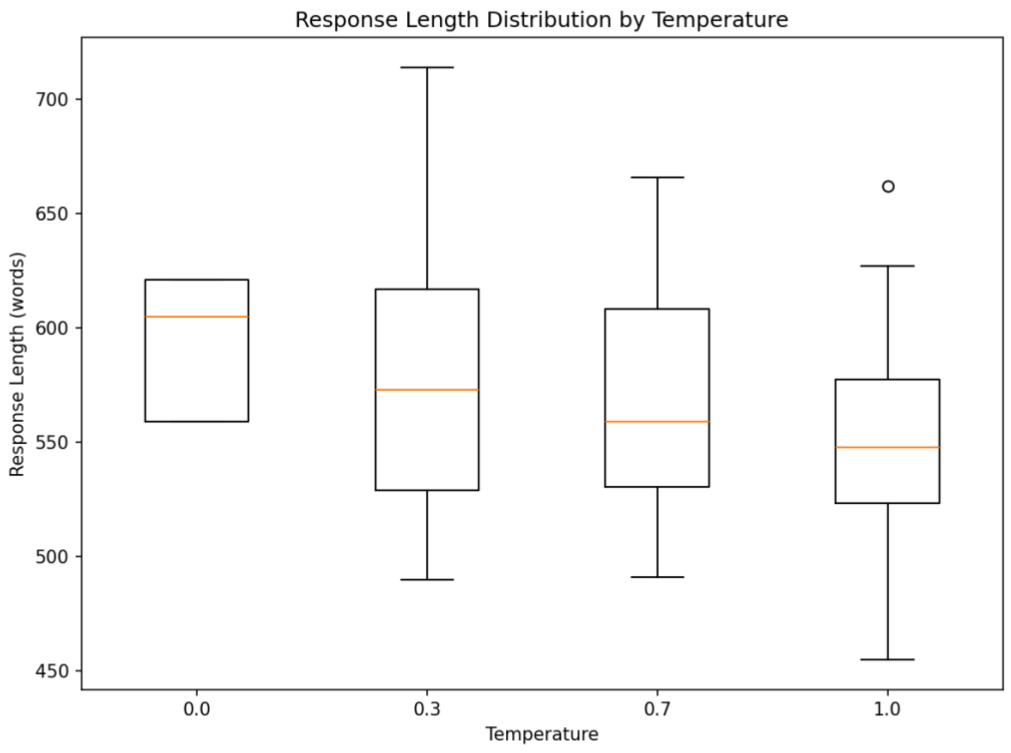

Fig 2: Boxplots to represent length distribution by temperature.

The similarity heatmap further supports this conclusion. Cosine similarity is highest within temperature 0.0 responses, confirming strong structural consistency. Similarity decreases progressively as temperature differences increase, with the lowest values observed between temperature 0.3 and 1.0. This indicates that higher temperature introduces semantic divergence and structural variability. Together, these analyses quantitatively confirm the theoretical expectation that increasing temperature increases entropy, reduces predictability, and lowers response similarity.

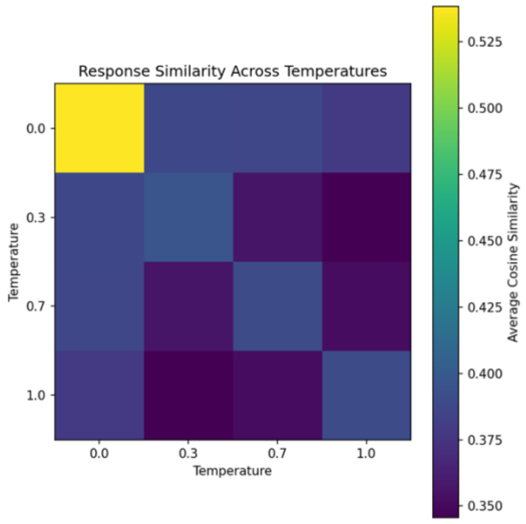

Fig 3: Heatmap representing the cosine similarity between responses at different temperatures.

7. Conclusion

This assignment made the relationship between sampling theory and practical API usage very concrete. The key findings were: temperature 0.0 produces fully deterministic output with zero variation across runs; temperature 0.3 introduces slight variation while maintaining strong structural consistency; temperature 0.7 produces meaningfully diverse outputs with good coherence; and temperature 1.0 maximizes creative variation while still maintaining relevance for well-designed prompts.

The additional similarity and length-distribution analyses further reinforced these conclusions quantitatively. The response similarity heatmap showed that temperature 0.0 had the highest intra-temperature cosine similarity, while similarity steadily decreased as temperature increased or as the gap between temperatures widened. This provided empirical evidence that higher temperature increases entropy and semantic divergence. The response length boxplots also revealed a clear pattern: lower temperatures produced longer and more tightly clustered outputs, while higher temperatures increased variance and structural variability. These visualizations confirmed that the theoretical effects of temperature scaling manifest measurably in real API outputs.

For practical applications, the recommended temperature settings are: T=0.0 for automated pipelines or tasks requiring reproducible outputs; T=0.3–0.5 for analytical, factual, or structured generation tasks; and T=0.7–1.0 for creative writing, brainstorming, and tasks where diversity is valued.

For future experiments, it would be worth testing temperatures above 1.0 to see where coherence starts to degrade, and testing the interaction between temperature and top-p simultaneously, since both affect the effective diversity of token selection. Combining them might produce different results than either alone. It would also be interesting to test the same temperature settings on a much larger model (such as LLaMA-3.1-70B) to see whether the temperature effect is more or less pronounced at larger scales.

Implementing the sampling functions from scratch was the most valuable part of the assignment. Going from "temperature makes output more random" to understanding exactly how dividing logits by a small number amplifies differences after exponentiation, and being able to visualize that directly in temperature_effect.png, gave a level of understanding that just reading the API documentation never would.

8. Reflection

Before this assignment, I had used temperature as a parameter in API calls without fully understanding what it was doing mechanically. I knew that low temperature meant more predictable and high temperature meant more random, but that description does not capture what is happening. Implementing softmax_with_temperature from scratch made it clear that temperature is simply a scalar division applied to the logit array before exponentiation. That simple operation has a profound effect: it stretches or compresses the gaps between logit values, which in turn determines how much one token dominates after softmax. Seeing this in the temperature_effect.png visualization alongside the actual numerical outputs made the mechanism intuitively and mathematically clear.

The extra credit analyses strengthened this understanding further. Computing cosine similarity across temperatures and analyzing response length distributions moved the conclusions beyond qualitative observation. Instead of simply noticing that higher temperatures "felt" more diverse, I was able to measure semantic divergence and structural variability directly. The similarity heatmap and boxplot visualizations made the entropy increase visible in a way that was both statistical and conceptual.

One result that surprised me was how completely identical the temperature 0.0 responses were across all five generations. Intellectually I understood that deterministic sampling always picks the argmax, but seeing five word-for-word identical 500-word responses in the JSON files made it viscerally obvious. It also made the value of slightly higher temperatures clear even when consistency is the goal. Temperature 0.3 produced outputs that were still highly consistent in structure and content but meaningfully different enough that running the generation multiple times had value.

The efficient top-k implementation using argpartition was also a good reminder that algorithmic complexity is not just a theoretical concern. The verification step comparing naive and efficient outputs confirmed they produce identical results, but the fact that argpartition achieves this in linear time while argsort requires O(V log V) is genuinely consequential at the scale of real language models. It is the kind of detail that is easy to overlook when working at the API level but becomes important when thinking about what is happening inside the model at inference time.