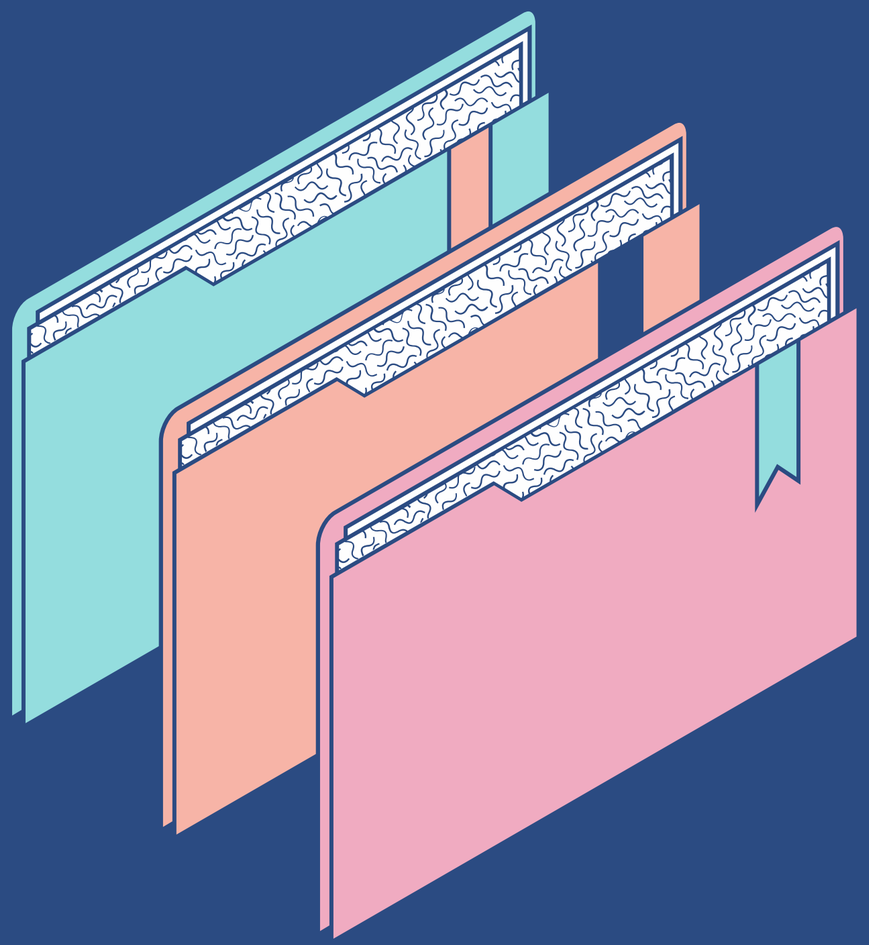
Sampling in Language Models

Arda Orcun

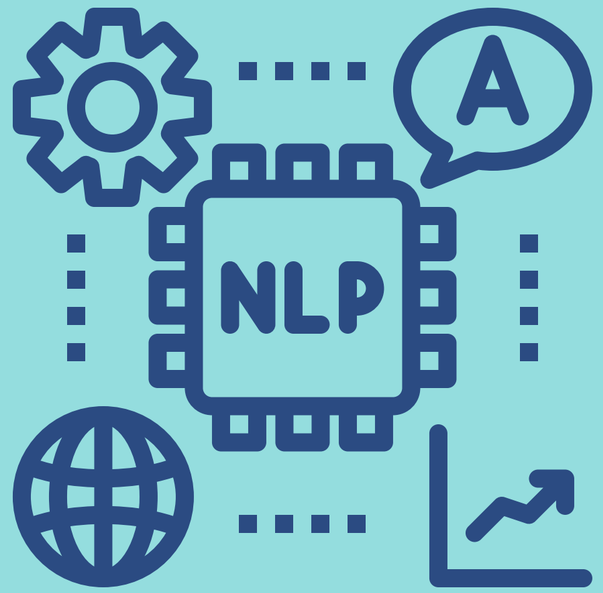


Overview

- Introduction to Language Models
- Importance of sampling in Language Models
- Sampling Technics in Language Models
- Top - K Sampling
- Hands on Top - K Sampling
- Conclusion

Introduction to Language Models

Language models are sophisticated algorithms designed to understand and generate human-like text, enabling machines to comprehend, predict, and produce natural language in a wide range of applications.



How do LMs work?

Language models employ tokens, representing words or subwords, to analyze patterns in text data. By processing sequences of tokens and learning from vast amounts of language data, these models utilize contextual information to predict the next word with remarkable accuracy.

But What is a Token?

In language models, tokens serve as the fundamental units of text representation, breaking down words or subwords into discrete elements for analysis. By encoding text into tokens, language models efficiently process and understand the underlying structure and context of language, enabling accurate predictions and generation of natural-sounding text.

In our exploration of language models, we'll dive into the technique of sampling tokens, a crucial process where we select individual tokens from a probability distribution generated by the model. This method allows us to explore different possibilities and generate diverse and contextually relevant text outputs.

Importance of Sampling in Language Models

Enhancing Diversity

Sampling enables language models to generate diverse outputs by exploring various possible sequences of tokens.

Facilitating Creativity

Sampling empowers language models to exhibit creativity and generate novel and unexpected outputs.

Mitigating Overfitting

By introducing randomness through sampling, language models can avoid overfitting to the training data and generalize better to unseen inputs.

Improving User Experience

Sampling contributes to enhancing the user experience by providing more varied and personalized interactions with language models.

Promoting Sustainability

Sampling in language models plays a crucial role in promoting sustainability by improving the efficiency of computational resources and reducing environmental impact. By employing sampling techniques, language models can optimize their computational processes, leading to more energy-efficient inference and training procedures.

Sampling Technics in Language Models

1) Greedy Search:

Greedy search selects the token with the highest probability at each step.

2) Beam Search:

Beam search maintains a fixed number of candidate sequences, known as the beam width, throughout the decoding process.

3) Temperature Sampling:

Temperature sampling introduces a temperature parameter to control the randomness of token selection.

4) Top-p Sampling:

Top-p sampling, also known as nucleus sampling, selects tokens from the smallest subset whose cumulative probability exceeds a predefined threshold.
It dynamically adjusts the token selection based on the probability distribution, promoting both diversity and coherence in generated text.

5)Top-k Sampling:

Top-k sampling limits token selection to the top-k most probable tokens at each step.
This method ensures diversity in generated text by considering a subset of the vocabulary distribution.
In our work, we will be interested in top-k sampling.

Top - K Sampling

Top-k sampling is a technique used in language models to generate diverse and contextually relevant text outputs. Instead of selecting tokens based solely on their individual probabilities, top-k sampling restricts token selection to the top-k most probable tokens at each step. By considering only a subset of the vocabulary distribution, top-k sampling promotes diversity in generated text while ensuring that selected tokens are still likely to occur in the given context. This method strikes a balance between exploration and exploitation, allowing language models to produce coherent and varied outputs across a wide range of natural language processing tasks.

Hands on Top - k Sampling

Dataset Used: "Open Platypus"

- **Description:** The "Open Platypus" dataset, commonly utilized for instruction fine-tuning, is a collection of datasets sourced from various repositories. It offers a diverse range of text data and is fully open-sourced, available for free use.
- **Release:** Boston University introduced the "Open Platypus" dataset in 2023. It serves as a valuable resource for training language models.
- **Accessibility:** Our data is sourced from HuggingFace, a platform renowned for hosting a wide array of text, image, and audio datasets. HuggingFace provides the latest datasets and models, all accessible at no cost and open-source.

Lee et al 2023: https://arxiv.org/abs/2308.07317

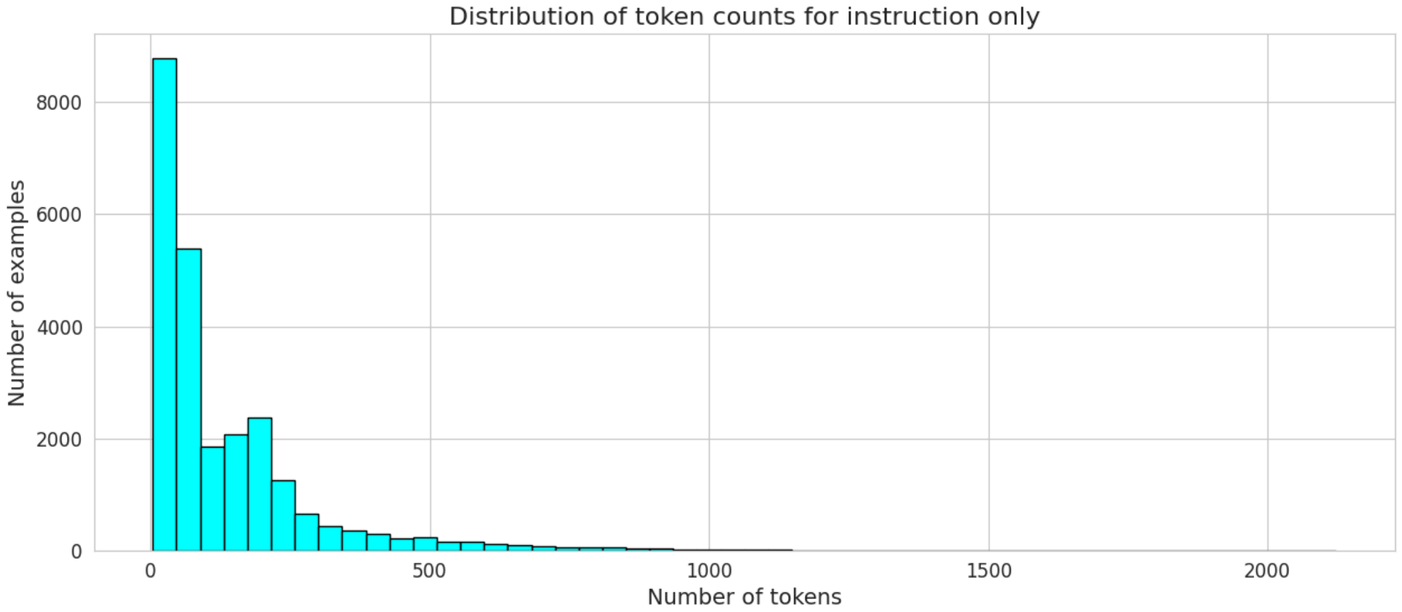
Hands on Top - k Sampling

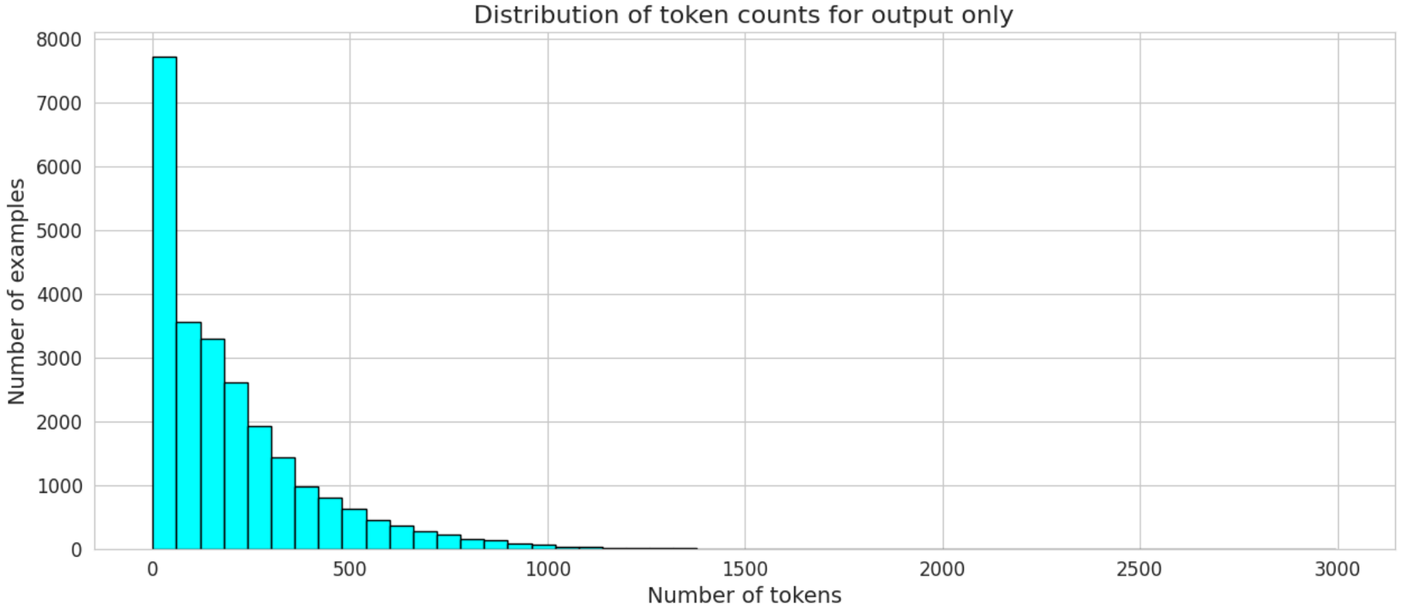
Language Model Used: "Llama 3"

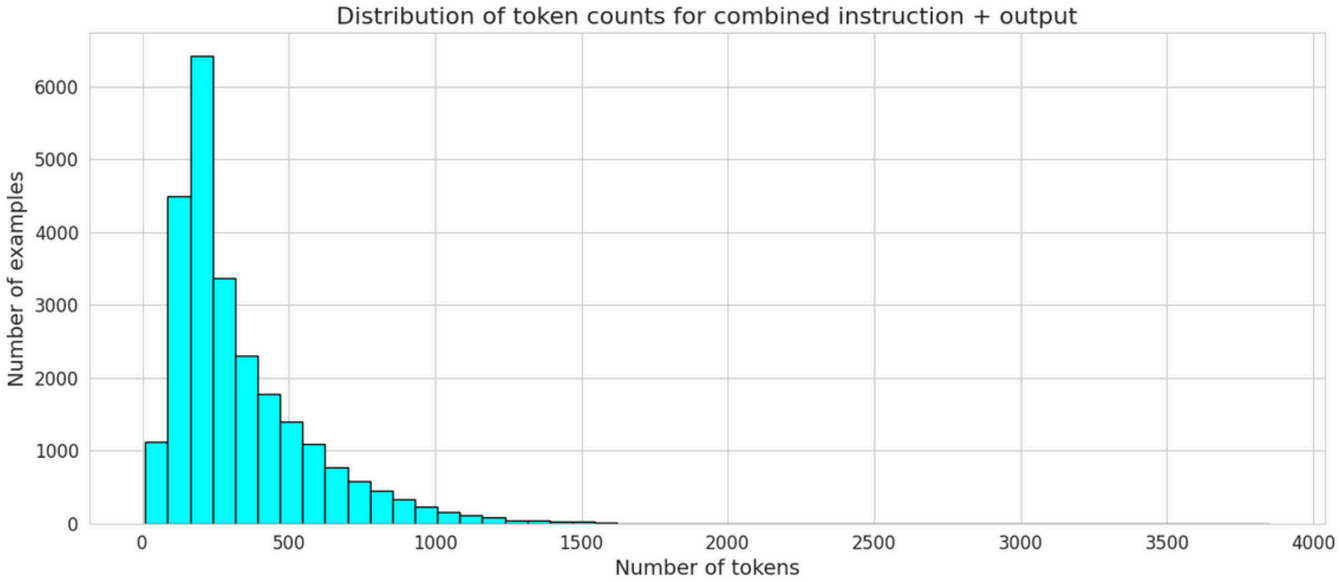
- Description: Our work utilizes the "Llama 3" Language Model (LLM), which stands as the State-of-the-Art (SoTA) and was released on April 18, 2024. Llama 3 is an open-source language model developed with 8B and 70B parameters, making it a powerful tool for natural language processing tasks.
- Training Data: Llama 3 was trained on an extensive dataset comprising 15 trillion tokens, enabling it to grasp intricate language patterns and nuances.
- Performance: As per the latest benchmark tests available at the time of this presentation's preparation, Llama 3 currently holds the title of the best Language Model (LLM) in the English language.
- Accessibility: The "Llama 3" model is accessible via HuggingFace.

Touvron et al 2023: https://arxiv.org/abs/2302.13971

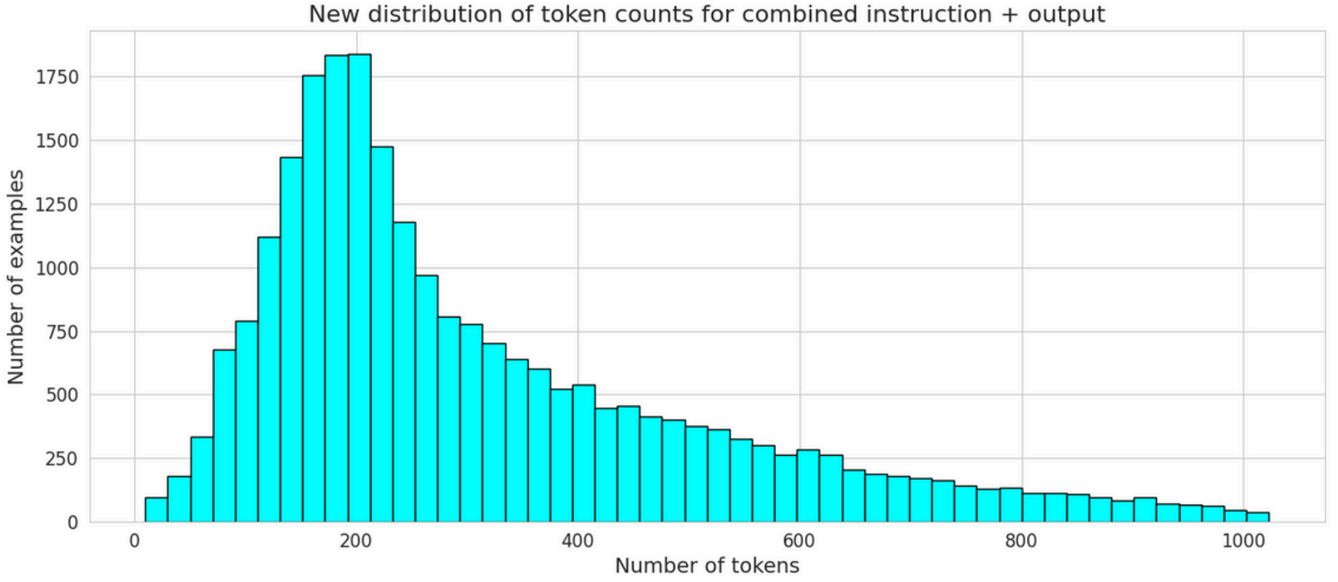
You can check the blog for Llama3: https://ai.meta.com/blog/meta-llama-3/


Distribution of token counts for instruction only
Number of examples
Number of tokens
0
2000
4000
6000
8000
0
500
1000
1500
2000


Distribution of token counts for output only
Number of examples
Number of tokens
0
1000
2000
3000
4000
5000
6000
7000
8000
0
500
1000
1500
2000
2500
3000



Removing tokens according to a threshold value (1024):



Hands on Top - k Sampling

Near-Duplicate Detection Algorithm
After token removal based on a threshold, we proceed with the near-duplicate detection process. This involves utilizing cosine similarity and nearest neighbor algorithms with an embedding model. The algorithm operates as follows:

- The function primarily employs cosine similarity to identify near-duplicates. This involves comparing the cosine similarity between each embedding and its nearest neighbor against a specified threshold.
- The actual similarity computation relies on cosine distance. This involves computing the cosine distance between two vectors, which is defined as the Euclidean distance between their corresponding components

```python
def deduplicate_dataset(dataset: Dataset, model: str, threshold: float):
    sentence_model = SentenceTransformer(model)
    outputs = [example["output"] for example in dataset['train']]

    print("Converting text to embeddings...")
    embeddings = sentence_model.encode(outputs, show_progress_bar=True)
    dimension = embeddings.shape[1]
    index = faiss.IndexFlatIP(dimension)
    normalized_embeddings = embeddings / np.linalg.norm(embeddings, axis=1, keepdims=True)
    index.add(normalized_embeddings)

    print("Filtering out near-duplicates...")
    D, I = index.search(normalized_embeddings, k=2)
    to_keep = []

    for i in tqdm(range(len(embeddings)), desc="Filtering"):
        # If the second closest vector (D[i, 1]) has cosine similarity above the threshold
        if D[i, 1] >= threshold:
            # Check if either the current item or its nearest neighbor is already in the to_keep list
            nearest_neighbor = I[i, 1]
            if i not in to_keep and nearest_neighbor not in to_keep:
                # If not, add the current item to the list
                to_keep.append(i)
        else:
            # If the similarity is below the threshold, always keep the current item
            to_keep.append(i)

    dataset = dataset['train'].select(to_keep)
    return DatasetDict({"train": dataset})
```

Li and Zhang et al 2023: https://arxiv.org/abs/2308.03281

Hands on Top - k Sampling

Following deduplication of our dataset with a rate of 0.95, we further refine our dataset using top-k sampling.

- The parameter for top-k sampling is set to k = 500.

Sampling Process:

- Deduplication Rate: A deduplication rate of 0.95 ensures that nearly identical examples are filtered out, enhancing dataset diversity and quality.
- Top-k Sampling: With k = 500, the top-k sampling method selects the 500 most probable examples at each step, ensuring a representative and high-quality dataset.

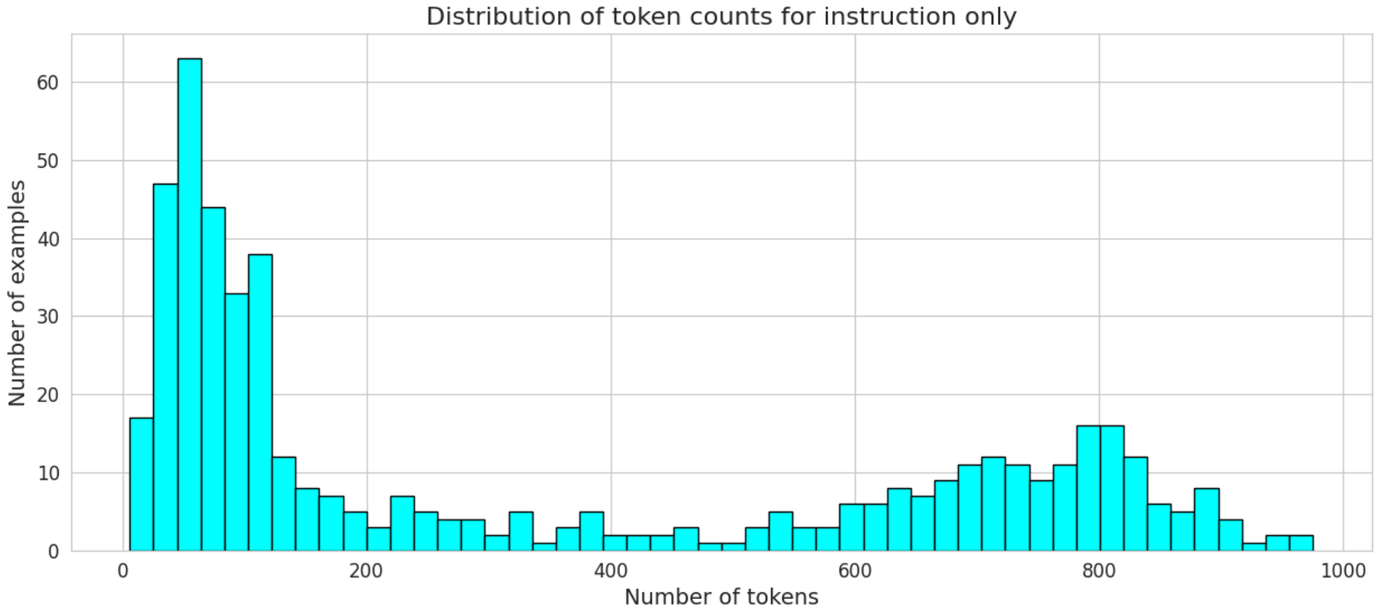
Results:

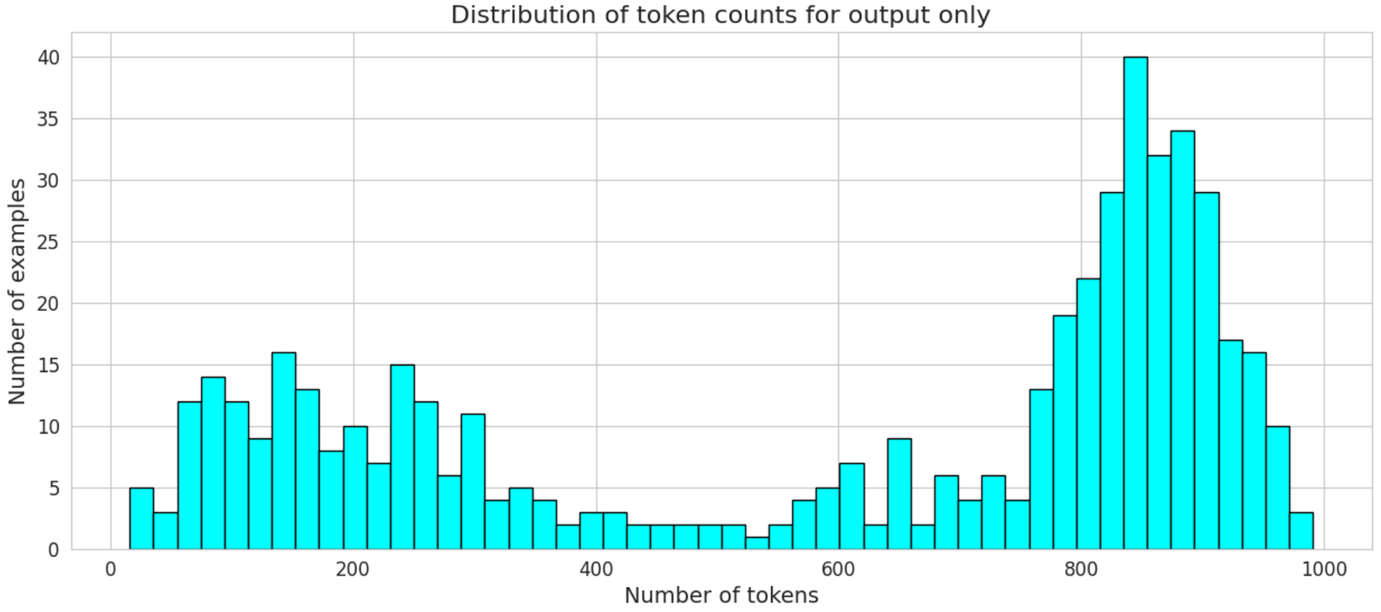
- Our sampling process has yielded a refined dataset that embodies both diversity and quality.
- By leveraging top-k sampling, we have curated a dataset that captures a wide range of linguistic patterns and contexts, essential for robust model training and evaluation.

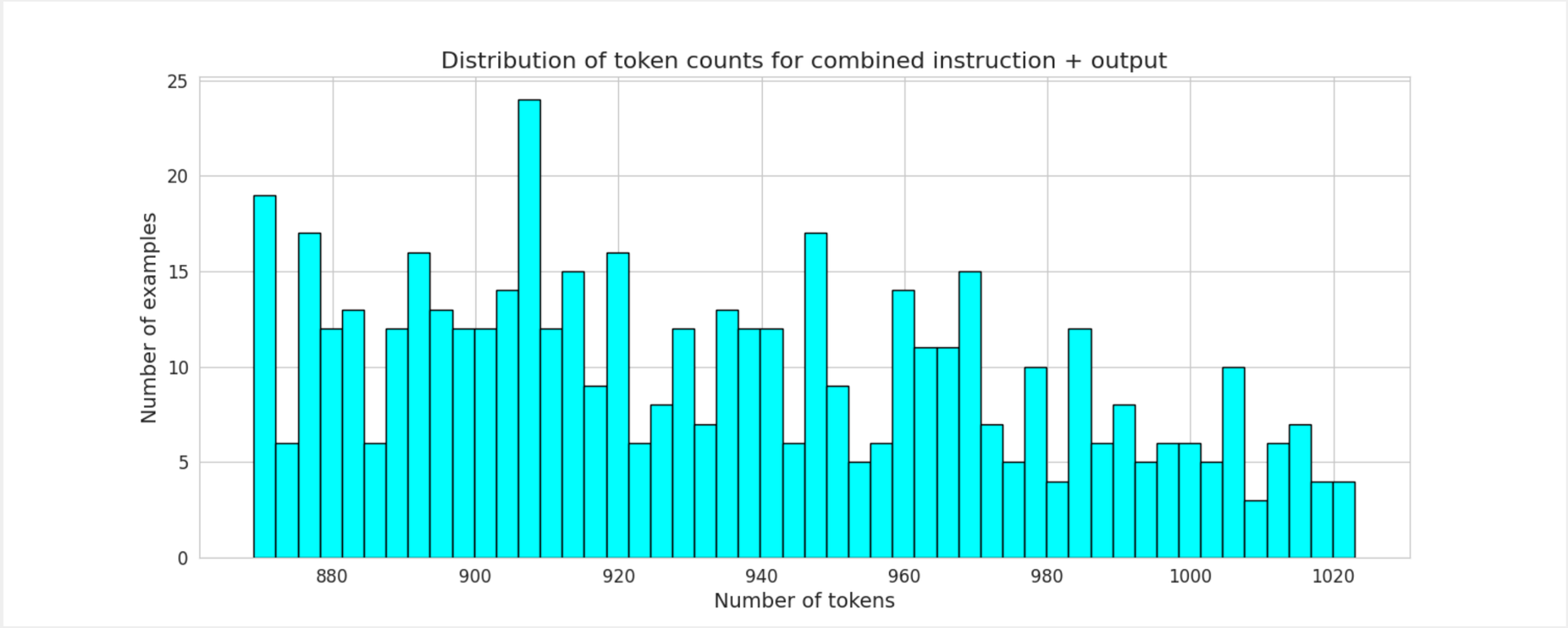
```python
# Get the top k rows with the most tokens
def get_top_k_rows(dataset, token_counts, k):
    # Sort by descending token count and get top k indices
    sorted_indices = sorted(range(len(token_counts)), key=lambda i: token_counts[i], reverse=True)
    top_k_indices = sorted_indices[:k]

    # Extract top k rows
    top_k_data = {
        "instruction": [dataset['train'][i]["instruction"] for i in top_k_indices],
        "output": [dataset['train'][i]["output"] for i in top_k_indices]
    }

    return Dataset.from_dict(top_k_data)
```


Distribution of token counts for instruction only
Number of examples
Number of tokens
0
10
20
30
40
50
60
0
200
400
600
800
1000


Distribution of token counts for output only
Number of examples
Number of tokens
0
5
10
15
20
25
30
35
40
0
200
400
600
800
1000


Distribution of token counts for combined instruction + output
Number of examples
0
5
10
15
20
25
880
900
920
940
960
980
1000
1020
Number of tokens

Fine - Tuning a LLM

To assess the numeric differences between datasets, we will leverage the "YTU-Cosmos" Language Model (LLM).
Our analysis will focus on evaluating training time and loss metrics to compare the performance of sampled datasets.
We will compare the performance of our original dataset against the refined dataset obtained through sampling techniques.
By analyzing training time and loss values, we aim to quantify the impact of sampling on dataset characteristics and model performance.
You can access YTU-Cosmos model from HuggingFace.

You can reach the model : https://huggingface.co/ytu-ce-cosmos/turkish-gpt2

Model Comparison: Sampled vs. Non-Sampled Datasets

- We fine-tuned two separate models: one with a sampled dataset and one with a non-sampled dataset.
- Both models were trained using identical parameters and for the same duration of 5 epochs with 2xT4 GPUs.

Training Metrics:

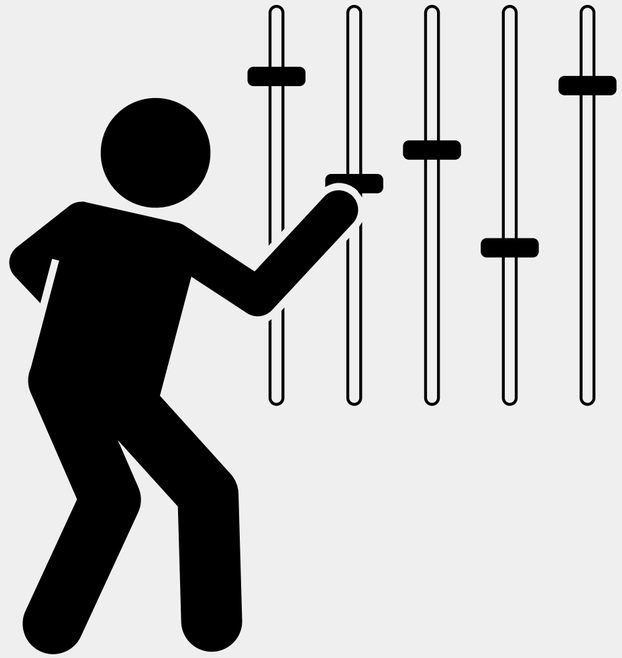
- Throughout the training process, we monitored key metrics including loss, GPU hour, GPU power, and others.

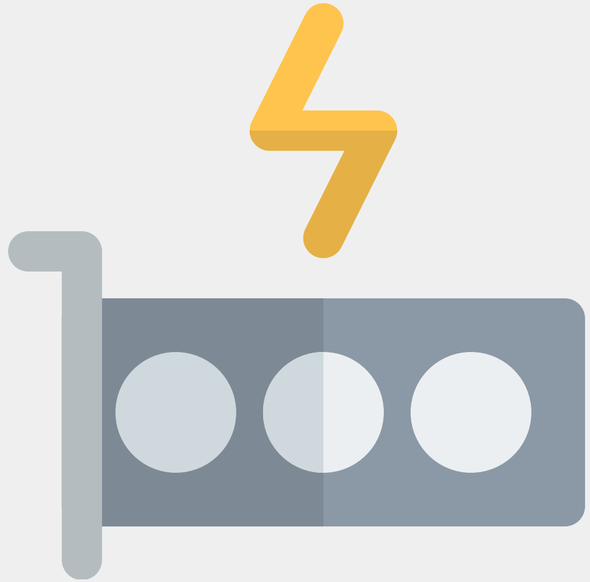
Understanding GPU Hour and GPU Power:

- GPU Hour: This metric represents the total time (in hours) that a GPU is actively utilized during model training. It accounts for both the duration and intensity of GPU usage.
- GPU Power: GPU power refers to the electrical power consumption of the GPU during training. It measures the rate at which energy is consumed by the GPU to perform computational tasks.

Next Steps:

- In the following slides, we will present the comparative results between models trained on sampled and non-sampled datasets, focusing on performance metrics such as loss, training time, and GPU utilization.





Conclusion

Key Observations:

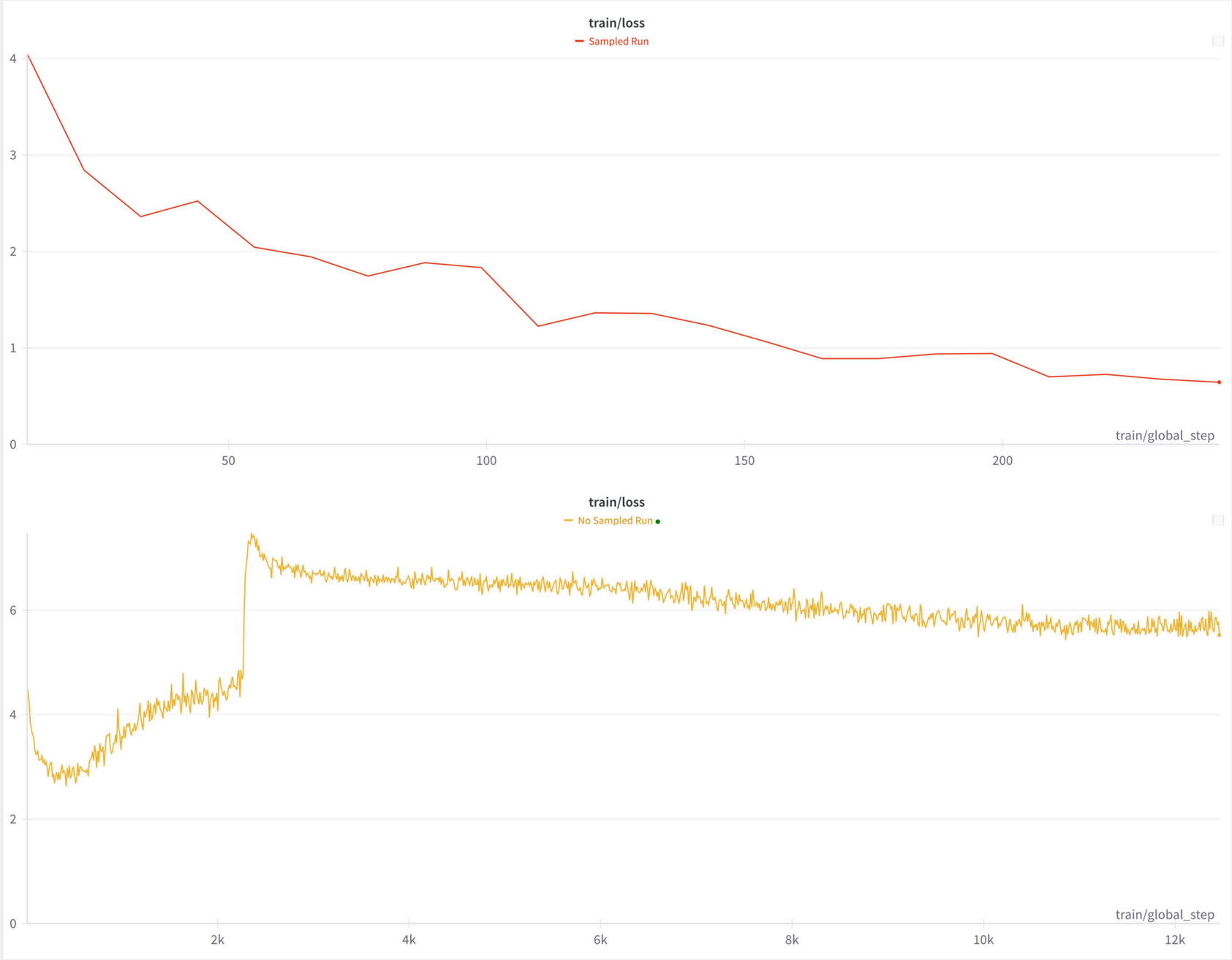
- After training both models, we observed significant differences in training time, loss, and GPU power utilization between the sampled and non-sampled datasets.

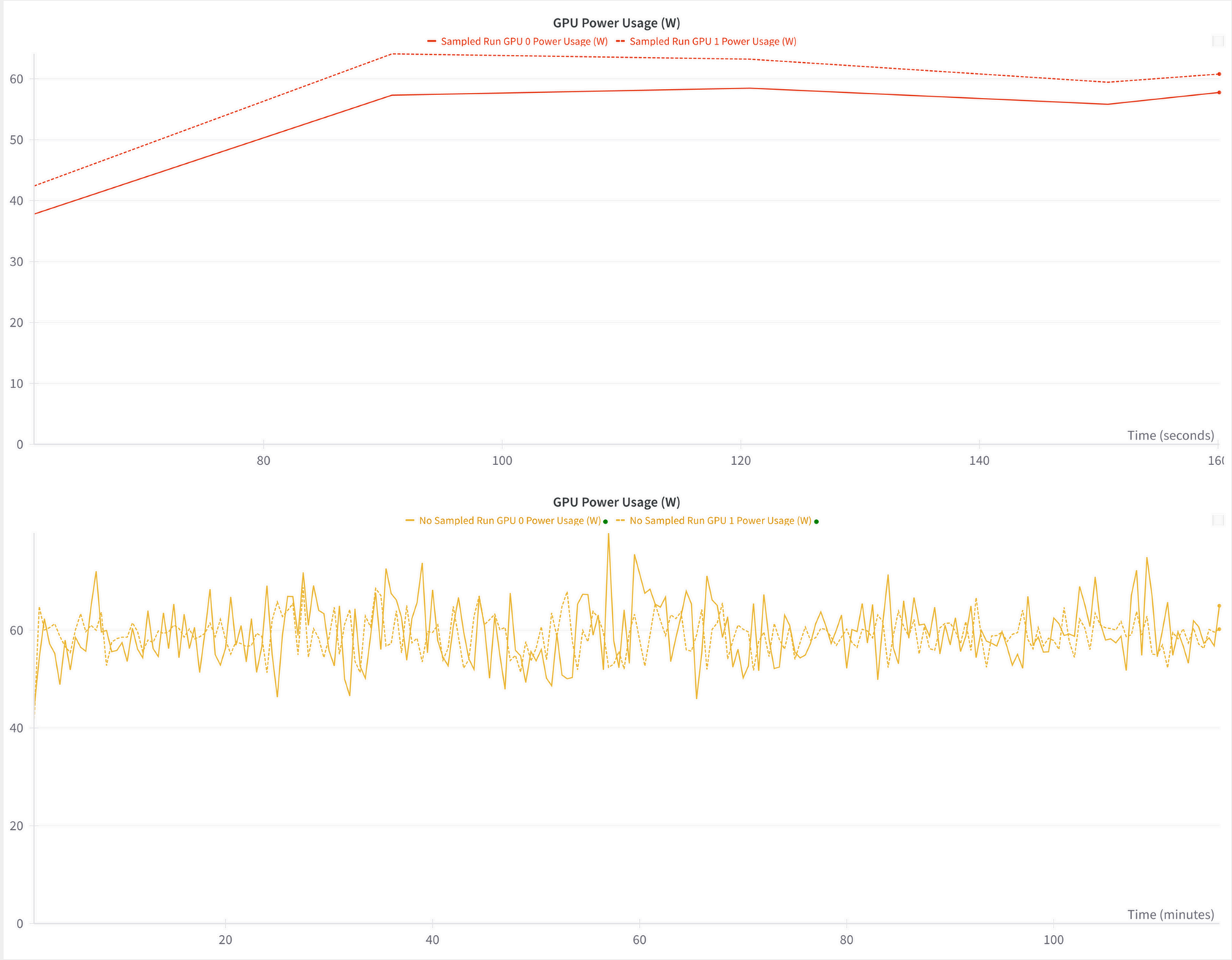
Results:

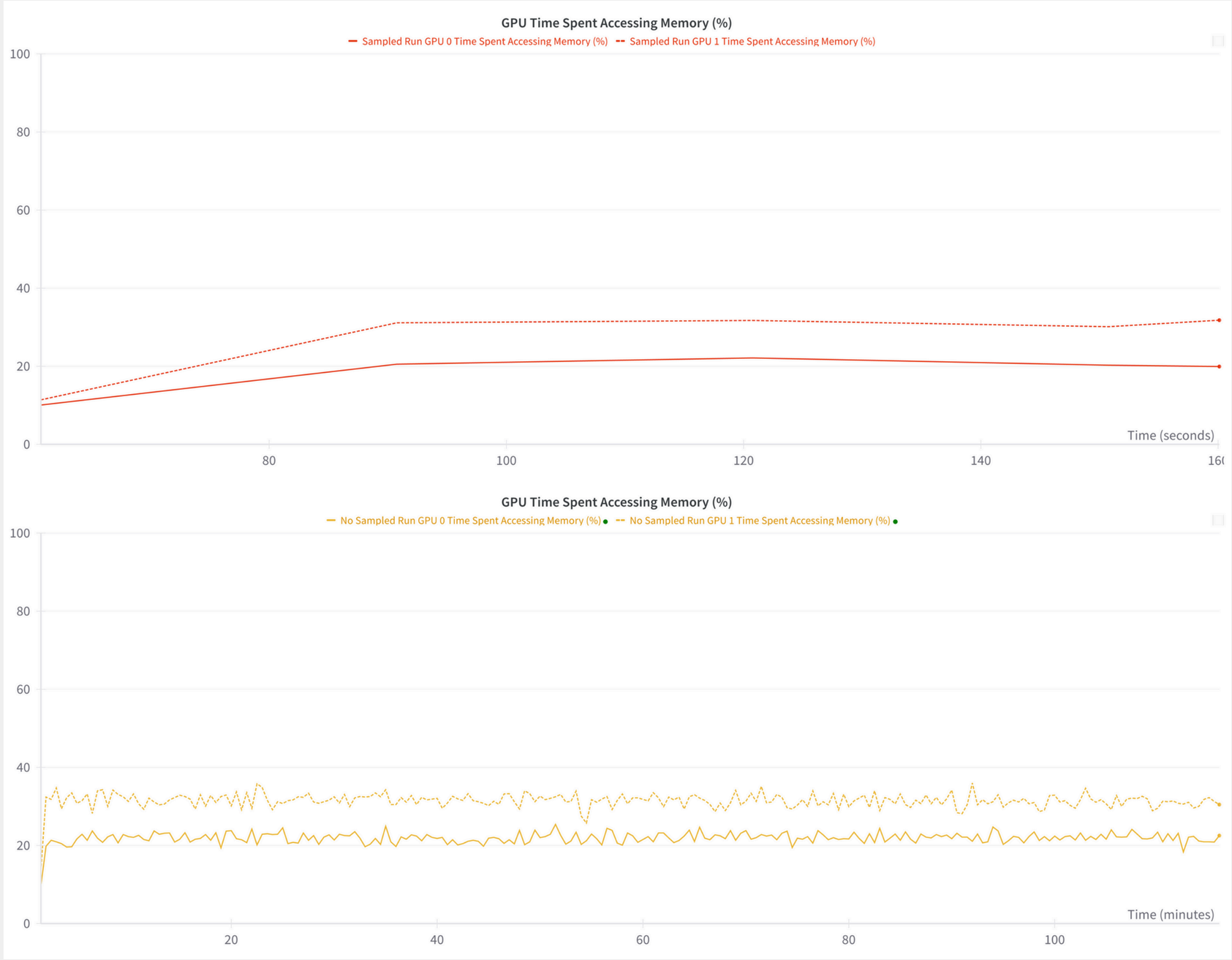
- **Training Time:**
  - Non-Sampled Dataset: 1 hour 55 minutes
  - Sampled Dataset: 2 minutes
- **Mean Loss:**
  - Non-Sampled Dataset: 6.72
  - Sampled Dataset: 0.8
- **Mean GPU Power Usage:**
  - Non-Sampled Dataset: 8050W
  - Sampled Dataset: 140W

Conclusion:

- The sampled training approach demonstrates superior performance in terms of training time efficiency, lower loss values, and reduced GPU power consumption.
- Based on these results, it is evident that the sampled training process not only enhances model performance but also promotes environmental sustainability through reduced energy consumption.


train/loss
Sampled Run
4
3
2
1
0
50
100
150
200
train/global_step
train/loss
No Sampled Run
6
4
2
0
2k
4k
6k
8k
10k
12k
train/global_step


GPU Power Usage (W)
Sampled Run GPU 0 Power Usage (W)
Sampled Run GPU 1 Power Usage (W)
60
50
40
30
20
10
0
Time (seconds)
80
100
120
140
GPU Power Usage (W)
No Sampled Run GPU 0 Power Usage (W)
No Sampled Run GPU 1 Power Usage (W)
60
40
20
0
Time (minutes)
20
40
60
80
100


GPU Time Spent Accessing Memory (%)
Sampled Run GPU 0 Time Spent Accessing Memory (%)
Sampled Run GPU 1 Time Spent Accessing Memory (%)
100
80
60
40
20
0
Time (seconds)
80
100
120
140
GPU Time Spent Accessing Memory (%)
No Sampled Run GPU 0 Time Spent Accessing Memory (%)
No Sampled Run GPU 1 Time Spent Accessing Memory (%)
100
80
60
40
20
0
Time (minutes)
20
40
60
80
100

Thanks for listening

All code segments are available at:
https://github.com/Stealeristaken/Sampling2Project