Amazon ML Challenge: Approach Summary

Prathamesh Gadekar Aditya Sanjay Nagulpelli Atharv Kurlapkar

September 16, 2024

1 Introduction

Our approach to the Amazon ML Challenge was shaped by the constraints of limited resources and time, given the 3-day duration of the competition. With access restricted to free versions of Kaggle and Google Colab, our team prioritized finding an efficient solution that leveraged pre-trained models to reduce computational overhead and achieve optimal results within the time limit.

2 Pre-trained Model Selection and Initial Experiments

We began by testing multiple multimodal models on a sample of the training data, typically in quantized form to manage computational resources. Our goal was to identify a pre-trained model that could offer strong initial performance and serve as a foundation for further fine-tuning.

After evaluating several candidates, the **Mini-CPM** model demonstrated the best performance on the sample data. It was chosen as the primary model for further processing due to its superior handling of the dataset and output quality compared to other alternatives. Also, due to computational limitations, quantized version of the model was chosen.

3 Strategy Development

At this point, we had two options:

1. Finetuning the Mini-CPM model to improve its output.
2. Post-processing the outputs of Mini-CPM to achieve the desired format.

Given the time constraints and resource limitations, we opted to focus on enhancing the output of Mini-CPM through prompt engineering, data engineering and post-processing techniques. We focused on designing better prompt for the model to get results very close to the desired format. Due to the high level of noise in the training data, fine-tuning was deliberately avoided to ensure the reliability of model's performance.

3.1 Parallel Processing via Distributed Resources

To maximize efficiency, we divided the dataset into smaller parts and ran parallel computations across multiple Kaggle accounts. Each team member processed a subset of the data, allowing us to work simultaneously and then merge the results into a unified output. This approach significantly reduced processing time.

3.2 Leveraging the Gemma 2 Model for Output Refinement

While the Mini-CPM model provided a solid starting point, we identified areas where the output required further refinement. At this stage, we introduced another language model, **Gemma 2**, into our pipeline. By engineering a custom prompt tailored to our requirements, we were able to improve the raw output generated by Mini-CPM.

As with the Mini-CPM processing, we split the data and ran computations in parallel using the free-tier resources available on Kaggle. Finally, we merged the refined outputs generated by Gemma 2.

3.3 Post-processing and Output Formatting

The outputs generated by the models were cleaned to remove inconsistencies and ensure alignment with the required format. We transformed the cleaned data into a CSV file, which served as the basis for further processing.

In the final step, the predictions contained in the CSV file were post-processed to meet the competition's specified format, ensuring our results were suitable for submission.

4 Methodology

We had access to a total of 20 GPUs from Kaggle. The approach consisted of the following steps:

1. **Divide** the `test.csv` file into 100 sequential segments.
2. **Process** the 100 segments in parallel, handling 20 at a time. This involved downloading images from the provided links, processing each image by submitting questions to a model, and saving the predictions along with their corresponding indices. This step took approximately 10 hours to complete.
3. **Merge** the results into a single file named `final_predictions.csv`.
4. **Split** the `final_predictions.csv` again into 20 sequential segments.
5. **Process** these 20 segments in parallel, sending the prediction values to the Gemma 2 model for further refinement and formatting. This step required about 2 hours to complete.
6. **Merge** the refined outputs once more and apply post-processing.

5 Folder Structure

Given below is the information about the files contained in the folder:

1. `data_analysis.py` – Performs data analysis on the training dataset.
2. `merge_files.py` – Merges all CSV files in a folder into one file.
3. `split_files.py` – Splits a single CSV file into a desired number of smaller files.
4. `amazon_ml_1.ipynb` – Implements the Mini-CPM model for visual question answering, used to predict the value of `entity_name`.
5. `amazon_llm_1.ipynb` – Implements the Gemma 2 model for further refining the output from the Mini-CPM model.
6. `post_processing.py` – Handles the post-processing of the output generated by the Gemma 2 model to produce the final submission file.

6 Conclusion

Our strategy for the Amazon ML Challenge relied heavily on selecting a strong pre-trained model, leveraging distributed computation to maximize available resources, and refining model outputs through a combination of prompt engineering and post-processing techniques. This approach enabled us to efficiently manage limited resources while delivering high-quality predictions within the competition's tight deadline.

Thank you to the Amazon ML Challenge team for organizing such an inspiring competition. It was a fantastic learning experience for our whole team!