LLaMA-Based Dataset Generation Details

Aryan-2201MC07

Contents

1 Model and Tokenizer Setup

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
import os
import json
import re


LLAMA_MODEL_PATH = "/scratch/nitishk_iitp/models"
# Load LLaMA tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(LLAMA_MODEL_PATH, local_files_only=True)
model = AutoModelForCausalLM.from_pretrained(
        LLAMA_MODEL_PATH,
        torch_dtype=torch.float16,
        device_map="auto",
        local_files_only=True
)
```

Listing 1: Load LLaMA Tokenizer and Model

Explanation: The tokenizer and model are loaded from a local path with auto device mapping and 16-bit floating precision for efficient inference.

2 Utility Functions

2.1 Split into Chunks

```
def split_into_chunks(lst, chunk_size):
    """Yield successive chunk_size-sized chunks from lst."""
    for i in range(0, len(lst), chunk_size):
        yield lst[i:i + chunk_size]
```

Listing 2: Split List into Chunks

Useful for batching operations over large datasets.

2.2 Extract Valid JSON

```
def extract_and_parse_json(text):
    """
    Extracts and parses a valid JSON object from the input text.
    Skips placeholder content (e.g., [...] or invalid JSON).
    """
    json_matches = re.finditer(r’\{.*?\}’, text, re.DOTALL)
    for match in json_matches:
        json_str = match.group(0)
        try:
            parsed = json.loads(json_str)
            if "candidates" in parsed and "ranking" in parsed:
                candidates = parsed["candidates"]
                ranking = parsed["ranking"]
                if isinstance(candidates, list) and isinstance(ranking, list) and all(candidates) and all(
     isinstance(r, int) for r in ranking):
                    return parsed
        except json.JSONDecodeError:
            # Skip invalid JSON and continue looking for valid JSON
            continue
    # If no valid JSON is found, print an error and return None
    print(" No valid JSON found in model output.")
    return None
```

Listing 3: Extract Valid JSON from Text

Ensures model responses are converted to usable structured data.

3 Sentence Ranking Using LLaMA

```python
def rank_sentences_llama(sentences, match_name, team_x, team_y, category):
    if not sentences:
        return {"candidates": [], "ranking": []}

    prompt = (
    f"You are a highly capable assistant tasked with helping sports journalists analyze match insights.\n"
    f"Match: {match_name}\n"
    f"Teams: {team_x} vs {team_y}\n"
    f"Insight Category: {category}\n\n"
    f"## Objective:\n"
    f"Evaluate the *new record-related sentences* from the match by ranking them uniquely based on the
     following *five criteria*, in order of importance:\n\n"
    f"1. *Sports relevance*  How strongly the sentence connects to the actual performance, statistics, or
     significant records of the match.\n"
    ...........
        f"## Sentences to rank:\n"
    )

    for i, sentence in enumerate(sentences, 1):
        prompt += f"{i}. {sentence}\n"

    prompt += (
        """

      1. *Output Format*:
       - Return only a valid JSON object with two keys: "candidates" and "ranking".
       Example format:{
                "candidates": [...],
                "ranking": [...]
                }
       - "candidates" must be a list of all input sentences in the same order they are provided.
        - "ranking" must be a list of distinct integers corresponding to the rank of each sentence in "
     candidates". For each sentences in candiadates provide corresponding ranking using the criteria provided
     above.
       .........
    """
    )

    inputs = tokenizer(prompt, return_tensors="pt", truncation=False).to(model.device)
    output = model.generate(
        **inputs,
        max_new_tokens=1024,
        do_sample=False,
        top_p=1.0,
        temperature=0.0,
        pad_token_id=tokenizer.eos_token_id,
    )
    generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
    print("\n Model output:")
    print("-" * 80)
    print(generated_text)
    print("-" * 80)
    parsed = extract_and_parse_json(generated_text)
    if not parsed:
        return {"candidates": [], "ranking": []}

    candidates = parsed.get("candidates", [])
    ranks = parsed.get("ranking", [])
    if not candidates or len(candidates) != len(ranks):
        print(" Parsed JSON inconsistent:", parsed)
        return {"candidates": [], "ranking": []}

    return {"candidates": candidates, "ranking": ranks}
```

Listing 4: Rank Sentences Using LLaMA Model

4 Processing JSON Files

```
def process_json_files(insights_dir, prefix, categories):
    combined = {category: [] for category in categories}
    for filename in os.listdir(insights_dir):
        if filename.startswith(prefix) and filename.endswith(".json"):
            file_path = os.path.join(insights_dir, filename)
            try:
                with open(file_path, "r", encoding="utf-8") as f:
                    data = json.load(f)
                    for category in combined:
                        if category in data and isinstance(data[category], list):
                            combined[category].extend(data[category])
            except Exception as e:
                print(f" Skipped file {filename}: {e}")
    return combined
```

Listing 5: Load Insight JSON Files

Merges sentence-level insight data across multiple files per category.

5 Main Processing Pipeline

```
def process_sport_matches(main_directory, sport_name, output_file_path):
    combined_categories = [
        "Relevancy",
        "New Records",
        "Key Match Events",
        "Pre-Game Insights",
        "Post-Match Reflections",
        "Miscellaneous Highlights",
        "Others"
    ]

    sport_path = os.path.join(main_directory, sport_name)
    if not os.path.isdir(sport_path):
        print(f" Invalid sport directory: {sport_path}")
        return

    with open(output_file_path, "w", encoding="utf-8") as output_file:
        match_dirs = os.listdir(sport_path)
        for match_folder in match_dirs:
            if match_folder == ".ipynb_checkpoints":
                continue

            match_path = os.path.join(sport_path, match_folder)
            if not os.path.isdir(match_path):
                continue

            try:
                match_info = re.match(
                    r"^(.*?)\s+v(?:\.|s)?\s+(.*?)\s+([a-zA-Z0-9]+)?\s*(\d{4}-\d{2}-\d{2})$",
                    match_folder, flags=re.IGNORECASE
                )
                if not match_info:
                    print(f" Skipped invalid match name: {match_folder}")
                    continue

                team_x, team_y, match_type, match_date = match_info.groups()
                match_type = match_type or ""
                match_name = f"{team_x} vs {team_y} {match_type} on {match_date}".strip()

                insights_path = os.path.join(match_path, "insights")
                if not os.path.exists(insights_path):
                    print(f" No insights folder in: {match_folder}")
                    continue

                print(f" Processing {match_name}")

                for category in combined_categories:
                    # First process AFTER
                    after_data = process_json_files(insights_path, "after", [category])
                    raw_sentences_after = after_data.get(category, [])
                    valid_sentences_after = [
                        s.strip() for s in raw_sentences_after
                        if s and s.strip().lower() not in {"relevant", "irrelevant"} and len(s.strip()) > 1
                    ]

                    if valid_sentences_after:
                        result_after = rank_sentences_llama(
                            valid_sentences_after, match_name, team_x, team_y, category
                        )
                        if result_after["candidates"]:
                            output_obj_after = {
                                "candidates": result_after["candidates"],
                                "ranking": result_after["ranking"]
                            }
                            output_file.write(json.dumps(output_obj_after, ensure_ascii=False) + "\n")

                    # Then process BEFORE
                    before_data = process_json_files(insights_path, "before", [category])
                    raw_sentences_before = before_data.get(category, [])
                    valid_sentences_before = [
                        s.strip() for s in raw_sentences_before
                        if s and s.strip().lower() not in {"relevant", "irrelevant"} and len(s.strip()) > 1
                    ]

                    if valid_sentences_before:
                        result_before = rank_sentences_llama(
                            valid_sentences_before, match_name, team_x, team_y, category
                        )
                        if result_before["candidates"]:
                            output_obj_before = {
                                "candidates": result_before["candidates"],
                                "ranking": result_before["ranking"]
                            }
                            output_file.write(json.dumps(output_obj_before, ensure_ascii=False) + "\n")

            except Exception as e:
                print(f" Error processing match ’{match_folder}’: {e}")

    print(f"\n Output written to: {output_file_path}")
```

Listing 6: Process Matches and Generate Ranked Outputs

This drives the full ranking pipeline per sport folder.

6 Execution Block

```
if __name__ == "__main__":
    main_dir = r"GPT4oFull"
    selected_sport = r"Odi"  #<--Change this to the sport directory you want
    output_jsonl = r"datasetOdi.jsonl"
    process_sport_matches(main_dir, selected_sport, output_jsonl)
```

Listing 7: Run the Full Pipeline

7 Prompt Template Used

```
prompt = (
f"You are a highly capable assistant tasked with helping sports journalists analyze match insights.\n"
f"Match: {match_name}\n"
f"Teams: {team_x} vs {team_y}\n"
f"Insight Category: {category}\n\n"
f"## Objective:\n"
f"Evaluate the *new record-related sentences* from the match by ranking them uniquely based on the
 following *five criteria*, in order of importance:\n\n"
f"1. *Sports relevance*  How strongly the sentence connects to the actual performance, statistics, or
 significant records of the match.\n"
f"2. *Emotional intensity*  Sentences that evoke strong feelings (e.g., pride, excitement, shock,
 disappointment) should be ranked higher.\n"
f"3. *Presence of sarcasm*  If any sentence subtly critiques or uses irony to emphasize a performance (
 positive or negative), it should gain weight for creativity.\n"
f"4. *Mentions of important people*  Sentences referencing star players, captains, or iconic figures
 should be prioritized over generic records.\n"
f"5. *Winning-related or buzz words*  Look for emotionally charged or impactful words like 'heroic', '
 dominant', 'smashed', 'historic', etc. These enhance audience appeal.\n\n"
f"The *higher* a sentence aligns with these criteria, the *better* its rank should be (i.e., rank 1 is the
  best). You must compare and evaluate *relatively*.\n"
f"No two sentences can share the same rank.\n\n"
    f"## Sentences to rank:\n"
)

for i, sentence in enumerate(sentences, 1):
    prompt += f"{i}. {sentence}\n"

prompt += (
    """

  1. *Output Format*:
   - Return only a valid JSON object with two keys: "candidates" and "ranking".
   Example format:{
            "candidates": [...],
            "ranking": [...]
            }
   - "candidates" must be a list of all input sentences in the same order they are provided.
   - "ranking" must be a list of distinct integers corresponding to the rank of each sentence in "
 candidates". For each sentences in candiadates provide corresponding ranking using the criteria provided
 above.
   - The length of "ranking" must exactly match the length of "candidates".
    Note: Don't sort the sentences based on rank
 2. *Validation Requirements*:
   - Ensure the JSON is syntactically valid:
     * No missing or extra brackets.
     * No missing or misplaced commas.
     * No trailing commas.
     * No missing or null values.
     * Properly escaped special characters if present.

 3. *Important Instructions*:
   - Do not include any explanation, commentary, or additional text outside the JSON object.
   - Make sure NO two sentences should have same rank
   - Verify that the output JSON adheres to the format requirements and contains no errors.

 4. *Common Mistakes to Avoid*:
   - Missing or extra brackets, commas, or values.
   - Mismatched lengths between "candidates" and "ranking".
   - Including invalid JSON, such as unescaped special characters or invalid key-value pairs.

Now, generate the output JSON as specified.
"""
)
```

8 Model Used

- **Model:** Llama-3.3-70B-Instruct
- **Library:** HuggingFace Transformers
- **Precision:** Float16
- **Device:** Auto-detected (e.g., CUDA/GPU)

article [a4paper,margin=1in]geometry array

9 Total Number of Data Samples Generated

Category	**Details**

Cricket (ODI, T20)	`datasetodi1 = [287]`
	`datasetodi2 = [58]`
	`datasetT20i = [880]`
	Cricket Total: [1225]

MLB	`datasetmlb1 = [197]`
	`datasetmlb2 = [269]`
	`datasetmlb3 = [413]`
	`datasetmlb4 = [118]`
	`datasetmlb5 = [190]`
	MLB Total: [1187]

NBA	`datasetnba1 = [228]`
	`datasetnba2 = [406]`
	`datasetnba3 = [236]`
	`datasetnba4 = [180]`
	`datasetnba5 = [208]`
	NBA Total: [1258]

Soccer	`datasetsoccer1 = [239]`
	`datasetsoccer2 = [158]`
	`datasetsoccer3 = [219]`
	`datasetsoccer4 = [149]`
	`datasetsoccer5 = [151]`
	`datasetsoccer6 = [70]`
	`datasetsoccer7 = [94]`
	Soccer Total: [1080]

Overall Total Number of Datasets: 4750