

✚ Getting started with CoAuthor

Goal: Download and read the CoAuthor dataset

Steps

1. Download CoAuthor
2. Read writing sessions
3. Examine events

> 1. Download CoAuthor

[] ↳ 2 cells hidden

> 2. Read writing sessions

Each writing session is saved as a `.jsonl` file in CoAuthor. It is a very convenient format to store events occurred in the writing session: each line is a JSON object, representing an *event*, and there are many lines in the file, representing *a sequence of events*.

Let's read one of the files and see what it looks like!

```
.jsonl is the JSON Lines text format. It is convenient for storing structured data that may be processed one record at a time.  
For more information, please refer to https://jsonlines.org/.
```

[] ↳ 2 cells hidden

> 3. Examine events

[] ↳ 5 cells hidden

> Helper Functions

[] ↳ 9 cells hidden

✚ Compute Coherence Score

Use a pre-trained language model like BERT to generate embeddings for the AI suggestion and the context.

```
from sentence_transformers import SentenceTransformer, util

def compute_coherence(suggestion, context):
    model = SentenceTransformer('all-MiniLM-L6-v2')

    embeddings = model.encode([suggestion, context])

    similarity = util.cos_sim(embeddings[0], embeddings[1])[0][0]

    return similarity
```

```
# Example usage
suggestion = "A feline rested on the mat"
context = "The cat sat on the mat"
coherence_score = compute_coherence(suggestion, context)
print(f"Coherence score: {coherence_score}")
```

```
🔗 /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
Coherence score: 0.6996868252754211
```

```

# Load checkpoint
checkpoint_data = load_checkpoint()
coherence_scores = checkpoint_data['coherence_scores']
processed_files = checkpoint_data['processed_files']
last_processed_file = checkpoint_data['last_processed_file']
last_processed_index = checkpoint_data['last_processed_index']

for session_path in paths:
    # Skip files that have already been fully processed
    if session_path in processed_files:
        print(f"Skipping already processed file: {session_path}")
        continue

    events = read_writing_session(session_path)
    all_suggestions = extract_suggestions(events)

    # If this is the last partially processed file, skip suggestions that have already been processed
    if session_path == last_processed_file:
        all_suggestions = all_suggestions[last_processed_index:]

    for i, suggestion in enumerate(all_suggestions):
        acceptance = suggestion.get("acceptance")
        context = get_last_sentences(reconstruct_current_doc(events[:suggestion.get("eventNum")])) # Use the current document as context
        suggestion_text = suggestion.get("suggestion")
        coherence_score = compute_coherence(suggestion_text, context)
        coherence_dict = {
            "acceptance": acceptance,
            "suggestion": suggestion_text,
            "context": context,
            "score": coherence_score}
        coherence_scores.append(coherence_dict)

    # Update checkpoint data
    checkpoint_data['coherence_scores'] = coherence_scores
    checkpoint_data['last_processed_file'] = session_path
    checkpoint_data['last_processed_index'] = i + 1 # Save the next index to process

    # Mark the file as fully processed
    processed_files.append(session_path)
    checkpoint_data['processed_files'] = processed_files
    checkpoint_data['last_processed_file'] = None
    checkpoint_data['last_processed_index'] = 0

    # Save checkpoint after finishing a file
    save_checkpoint(checkpoint_data)

    print("Finished:", len(processed_files)/len(paths)*100, "%")

# Final save
save_checkpoint(checkpoint_data)

```

 Show hidden output

▼ Analysis


Calculate the average coherence score.

```

sum = 0
for d in coherence_scores:
    sum += d['score']
average_coherence = sum / len(coherence_scores)

print([d['score'] for d in coherence_scores])
# print(coherence_scores)
print(f"Average coherence score: {average_coherence}")

```

 [tensor(0.3937), tensor(0.3659), tensor(0.2648), tensor(0.2872), tensor(0.2380), tensor(0.2837), tensor(0.3490), tensor(0.2019), tensor(0.3786083161830902)]
Average coherence score: 0.3786083161830902

Calculate the average coherence scores for accepted suggestions and for rejected suggestions.

```
accepted_sum = 0
accepted_count = 0
rejected_sum = 0
rejected_count = 0

for d in coherence_scores:
    if d['acceptance'] == 'accepted':
        accepted_sum += d['score']
        accepted_count += 1
    else:
        rejected_sum += d['score']
        rejected_count += 1

average_accepted_coherence = accepted_sum / accepted_count
average_rejected_coherence = rejected_sum / rejected_count

print(f"Average coherence score for accepted suggestions: {average_accepted_coherence}")
print(f"Average coherence score for rejected suggestions: {average_rejected_coherence}")
```

```
🔗 Average coherence score for accepted suggestions: 0.38358813524246216
Average coherence score for rejected suggestions: 0.36347082257270813
```