# 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

```
!wget https://cs.stanford.edu/~minalee/zip/chi2022-coauthor-v1.0.zip
!unzip -q chi2022-coauthor-v1.0.zip
!rm chi2022-coauthor-v1.0.zip
```

```
--2025-02-27 03:06:50--  https://cs.stanford.edu/~minalee/zip/chi2022-coauthor-v1.0.zip
Resolving cs.stanford.edu (cs.stanford.edu)... 171.64.64.64
Connecting to cs.stanford.edu (cs.stanford.edu)|171.64.64.64|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 49956179 (48M) [application/zip]
Saving to: ‘chi2022-coauthor-v1.0.zip’

chi2022-coauthor-v1 100%[===================>]  47.64M  11.3MB/s    in 4.2s

2025-02-27 03:06:54 (11.3 MB/s) - ‘chi2022-coauthor-v1.0.zip’ saved [49956179/49956179]

replace coauthor-v1.0/e0435f4cf6fc435c872ffc5b66b66b0c.jsonl? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
```

```
import os

dataset_dir = './coauthor-v1.0'
paths = [
    os.path.join(dataset_dir, path)
    for path in os.listdir(dataset_dir)
    if path.endswith('jsonl')
]

print(f'Successfully downloaded {len(paths)} writing sessions in CoAuthor!')
```

```
Successfully downloaded 1447 writing sessions in CoAuthor!
```

## 2. Read writing sessions

Each writing session is saved as a `.jsonl` file in CoAuthor. It is a very convenient format to store events occured 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/.

```
import json

def read_writing_session(path):
    events = []
    with open(path, 'r') as f:
        for event in f:
            events.append(json.loads(event))
    print(f'Successfully read {len(events)} events in a writing session from {path}')
    return events
```

```
events = read_writing_session(paths[0])
```

```
Successfully read 2405 events in a writing session from ./coauthor-v1.0/608e5b73341a4f3ca937316f99dae32d.jsonl
```

## 3. Examine events

Whenever writers insert or delete text, move a cursor forward or backward, get suggestions from the system by pressing the tab key, or accept or dismiss suggestions, it is recorded as an event.

> Here is a list of all possible events in CoAuthor:
>
> - `system-initialize`
> - `text-insert`
> - `text-delete`
> - `cursor-forward`
> - `cursor-backward`
> - `cursor-select`
> - `suggestion-get`
> - `suggestion-open`
> - `suggestion-up`
> - `suggestion-down`
> - `suggestion-select`
> - `suggestion-close`

For more details, please check out our paper (Table 1): https://arxiv.org/pdf/2201.06796.pdf

```
event_names = [event['eventName'] for event in events]
event_names[:15]
```

```
['system-initialize',
 'text-insert',
 'text-insert',
 'text-delete',
 'text-delete',
 'text-insert',
 'text-insert',
 'text-insert',
 'text-insert',
 'text-insert',
 'text-insert',
 'text-insert',
 'text-insert',
 'text-insert',
 'text-insert']
```

Let's look at each *event* more closely now!

In the beginning of a writing session, you will see something like this for a system-initialize event:

```
events[0]
```

```
{'eventName': 'system-initialize',
 'eventSource': 'api',
 'eventTimestamp': 1630340187362,
 'textDelta': '',
 'cursorRange': '',
 'currentDoc': "Are We Being Bad Citizens If We Don't Keep Up With the News?\n\nIn your opinion, are we being bad citizens if we don't
keep up with the news? Do you think all people have some responsibility to know what is going on in the world? Does engaging with
current events actually do anything at all? Why do you think the way you do?\n\n---\n\nIn my opinion,\n",
 'currentCursor': 344,
 'currentSuggestions': [],
 'currentSuggestionIndex': 0,
 'currentHoverIndex': '',
 'currentN': '5',
 'currentMaxToken': '30',
 'currentTemperature': '0.2',
 'currentTopP': '1',
 'currentPresencePenalty': '0',
 'currentFrequencyPenalty': '0.5',
 'eventNum': 0}
```

Concretely, an *event* is a tuple of event name, timestamp, and snapshot of the current editor. This is designed to preserve every detail about interactions at a keystroke-level, so it is quite detailed as you can see!

> **Event and its metadata**

- `eventName`: event name (e.g. `system-initialize`)
- `eventSource`: event source (e.g. `user` or `api`)
- `textDelta`: text that has been changed compared to the previous event (if no change, empty)
- `cursorRange`: cursor location or selection that has been changed compared to the previous event (if no change, empty)

**Timestamp**

- `eventTimestamp`: timestamp of the event
- `eventNum`: index of the event

**Snapshot of the current editor**

- Editor
  - `currentDoc`: a writing prompt to start with (otherwise, empty)
  - `currentCursor`: cursor location
  - `currentSuggestions`: most recent suggestions that are stored and can be reopened
- Decoding parameters
  - `currentN`: the number of suggestions to generate per query (e.g. 5)
  - `currentMaxToken`: the maximum number of tokens to generate per suggestion
  - `currentTemperature`: sampling temperature to use for generation; higher values means the model will take more risks
  - `currentTopP`: nucleus sampling; the model considers the results of the tokens with top_p probability mass
  - `currentPresencePenalty`: positive values penalize new tokens based on whether they appear in the text so far, increasing the model's likelihood to talk about new topics
  - `currentFrequencyPenalty`: positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim

For more details on decoding parameters, please refer to https://beta.openai.com/docs/api-reference/completions.

## Helper Functions

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Reconstruct the current document.

```
def reconstruct_current_doc(events):
    current_doc = events[0]['currentDoc']  # Initialize with the prompt
    i = 1

    for event in events[1:]:  # Skip the first event (system-initialize)
        if event['eventName'] == 'text-insert' or event['eventName'] == 'text-delete':
            if any('retain' in d for d in event['textDelta'].get('ops')):
                retain_lst = [d.get('retain') for d in event['textDelta'].get('ops') if 'retain' in d]
                cursor_position = retain_lst[0]
            else:
                retain_lst = []
                cursor_position = 0

            if any('delete' in d for d in event['textDelta'].get('ops')):
                nums = [d.get('delete') for d in event['textDelta'].get('ops') if 'delete' in d]
                num_to_delete = 0
                for i in nums:
                    num_to_delete = num_to_delete + i
                current_doc = current_doc[:cursor_position] + current_doc[cursor_position+num_to_delete:] # Delete the text from the document

            if any('insert' in d for d in event['textDelta'].get('ops')):
                text_to_insert = [d.get('insert') for d in event['textDelta'].get('ops') if 'insert' in d][0] # Extract the text to insert fr
                if type(text_to_insert) != str:
                    text_to_insert = " "
                if len(retain_lst) > 1:
                    position_adj = 1
                else:
                    position_adj = 0
            current_doc = current_doc[:cursor_position+position_adj] + text_to_insert + current_doc[cursor_position+position_adj:] # Inse
    return current_doc
```

```
# Reconstruct the currentDoc
current_doc = reconstruct_current_doc(events)

# Print the final document
print("Final Document:")
print(current_doc)
```

```
Final Document:
Are We Being Bad Citizens If We Don't Keep Up With the News?

In your opinion, are we being bad citizens if we don't keep up with the news? Do you think all people have some responsibility to know w

---

In my opinion, it is the responsibility of a good citizen to keep up with the news.   I think it is important to know what is going on i

Keeping up with the news also allows the public to stay informed about issues such as climate change.  We are far behind when it comes t

When you have areas of the world that are experiencing historical high water levels due to climate change, there are often news reports

I believe that people who go through life oblivious to the problems that surround them are not good people.  The news often focuses on p
```

Extract the last three sentences from the document.

```
import re

def get_last_sentences(paragraph):
    """
    Extracts the last sentence from a paragraph.

    Args:
        paragraph: The input paragraph as a string.

    Returns:
        The last sentence of the paragraph, or None if no sentence is found.
    """
    if not paragraph:
        return None

    sentences = re.split(r'(?<!\w\.\w)(?<![A-Z][a-z]\.)(?<=\.|\?|!)\s', paragraph)

    sentences = [s.strip() for s in sentences if s.strip()]
    last_three = sentences[-3:]
    return " ".join(last_three)

# Example usage
text = "This is the first sentence. Here is the second sentence.\n\nAnd this is the last sentence! But what about this one?"
last_sentence = get_last_sentences(text)
print(last_sentence)

text_2 = "This is a paragraph without multiple sentences."
last_sentence_2 = get_last_sentences(text_2)
print(last_sentence_2)

text_3 = ""
last_sentence_3 = get_last_sentences(text_3)
print(last_sentence_3)
```

```
Here is the second sentence. And this is the last sentence! But what about this one?
This is a paragraph without multiple sentences.
None
```

Extract the acceptance status of each suggestion that the AI opens.

```
def extract_suggestions(events):
    """Track if a suggestion was accepted or rejected."""
    ai_suggestions = []  # Stores dicts with 'accepted' or 'rejected' and the suggestion
    last_suggestion_open = None  # Store the most recent `suggestion-open`

    for event in events:
        if event["eventName"] == "suggestion-open" and event.get("currentSuggestions"):
            last_suggestion_open = event  # Save the latest `suggestion-open`

        elif event["eventName"] == "suggestion-select" and last_suggestion_open:
            # Suggestion selected
            selected_index = event.get("currentSuggestionIndex", -1)

            # Ensure selected index is valid
            if 0 <= selected_index < len(last_suggestion_open["currentSuggestions"]):
                selected_suggestion = last_suggestion_open["currentSuggestions"][selected_index]["trimmed"]
                ai_suggestions.append({'acceptance':'accepted', 'eventNum':event["eventNum"], 'suggestion':selected_suggestion}) # Add sugge

            last_suggestion_open = None  # Reset, since we handled the acceptance

        elif event["eventName"] == "suggestion-close" and last_suggestion_open:
            # Suggestion closed without selection (rejected)
            rejected_suggestion = last_suggestion_open["currentSuggestions"][0]["trimmed"]
            ai_suggestions.append({'acceptance':'rejected', 'eventNum':event["eventNum"], 'suggestion':rejected_suggestion}) # Add suggestic
            last_suggestion_open = None  # Reset, since we handled the rejection

    return ai_suggestions

# Process all valid events
all_suggestions = extract_suggestions(events)

print(f"Total AI Suggestions: {len(all_suggestions)}")
print("Example AI Suggestion:", all_suggestions)
```

```
Total AI Suggestions: 4
Example AI Suggestion: [{'acceptance': 'accepted', 'eventNum': 95, 'suggestion': 'I think it is important to know what is going on in th
```

Functions for saving checkpoints so that progress is not lost if runtime disconnects.

```
import pickle

def save_checkpoint(checkpoint_data, checkpoint_file='/content/drive/MyDrive/checkpoint.pkl'):
    with open(checkpoint_file, 'wb') as f:
        pickle.dump(checkpoint_data, f)
    print(f"Checkpoint saved to {checkpoint_file}")

def load_checkpoint(checkpoint_file='/content/drive/MyDrive/checkpoint.pkl'):
    try:
        with open(checkpoint_file, 'rb') as f:
            checkpoint_data = pickle.load(f)
        print(f"Checkpoint loaded from {checkpoint_file}")
        return checkpoint_data
    except FileNotFoundError:
        print("No checkpoint found. Starting from scratch.")
        return {
            'coherence_scores': [],
            'processed_files': [],
            'last_processed_file': None,
            'last_processed_index': 0
        }
```

## 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}")
```

```
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)
```

```
Skipping already processed file: ./coauthor-v1.0/0837a09ef77a4f20b3a341a47276288b.jsonl
Skipping already processed file: ./coauthor-v1.0/d4a9f1685bf541d29e96abe560ef4589.jsonl
Skipping already processed file: ./coauthor-v1.0/e4de659d6e5a44eca6c88b18dc300d70.jsonl
Skipping already processed file: ./coauthor-v1.0/36a7437e63c64ad99a8e4cc9433a7be1.jsonl
Skipping already processed file: ./coauthor-v1.0/b6620ed18c174f708138abb59ade11ca.jsonl
Skipping already processed file: ./coauthor-v1.0/b3a1b8cfb7d94eb3a615600d440a0bc1.jsonl
Skipping already processed file: ./coauthor-v1.0/b47d3c5973494f02af1668b926a7210e.jsonl
Skipping already processed file: ./coauthor-v1.0/aea2e89578b3408983027d4f02ac76e6.jsonl
Skipping already processed file: ./coauthor-v1.0/af58ab552c724ef2bff4dd8108581e1c.jsonl
Skipping already processed file: ./coauthor-v1.0/5b7f010672c4459e86b69d0fc99f0509.jsonl
Skipping already processed file: ./coauthor-v1.0/543ec3b66b0c47269e9737a650da21b5.jsonl
Skipping already processed file: ./coauthor-v1.0/69655d09325f45e6bc4dbe10afdad674.jsonl
Skipping already processed file: ./coauthor-v1.0/2b0bf00c87074030b36a423d292605e7.jsonl
Skipping already processed file: ./coauthor-v1.0/7b3d6b7a25a94d4f85060bbee5dd4682.jsonl
Skipping already processed file: ./coauthor-v1.0/e3d026e8ecb1421ca9ef198526fd4633.jsonl
Skipping already processed file: ./coauthor-v1.0/3f08232e93af452cb4f47c0bc22a6980.jsonl
Skipping already processed file: ./coauthor-v1.0/478d7bee3c53454fb80f423138913c4f.jsonl
Skipping already processed file: ./coauthor-v1.0/95b8a64db3d3471f9662d0019357059d.jsonl
Skipping already processed file: ./coauthor-v1.0/ef145d5c1ad94981bbea37ec48c4eb2e.jsonl
Skipping already processed file: ./coauthor-v1.0/c6bf0514feec40d58281a3c276051cf0.jsonl
Skipping already processed file: ./coauthor-v1.0/d7e61761eda64607b4b1232dcc09bddc.jsonl
Skipping already processed file: ./coauthor-v1.0/beb2455e6a514e41a535aa51532f3d1f.jsonl
Skipping already processed file: ./coauthor-v1.0/25ffdce6009a47f08c72347a283d256e.jsonl
Skipping already processed file: ./coauthor-v1.0/30786a82a19c45a6a77a12963fa21ea0.jsonl
Skipping already processed file: ./coauthor-v1.0/786e6c194f0a4fd182440d297bfc3b9f.jsonl
Skipping already processed file: ./coauthor-v1.0/2b26596dffdc4403b160b0e12bc5daa9.jsonl
Skipping already processed file: ./coauthor-v1.0/06a646f4746648eebec3e1869b713d7e.jsonl
Skipping already processed file: ./coauthor-v1.0/758b76b82ccf413aa93fd4d74624d288.jsonl
Skipping already processed file: ./coauthor-v1.0/cc53a2225e9b413e84efba8500f8c907.jsonl
Skipping already processed file: ./coauthor-v1.0/02aa8f6a107341ceb9cd2ef1a43da832.jsonl
Skipping already processed file: ./coauthor-v1.0/820a0ffa457343b2a3d1e69a48d604e8.jsonl
Skipping already processed file: ./coauthor-v1.0/a7518c3553b7485a9e01e8bd95843abc.jsonl
Skipping already processed file: ./coauthor-v1.0/b4134879f6494953bfca7456a0bc8341.jsonl
Skipping already processed file: ./coauthor-v1.0/6a0e4a84b6624948b9373a84488399cd.jsonl
Skipping already processed file: ./coauthor-v1.0/ac0d02d15e164314841ae2ce49d2b2c9.jsonl
Skipping already processed file: ./coauthor-v1.0/e829b10c37c44068b58804a38e1f4840.jsonl
Skipping already processed file: ./coauthor-v1.0/3b62a2036d2643faa4590b94c3fb0d35.jsonl
Skipping already processed file: ./coauthor-v1.0/eecd784a495648f4a5b160beba42dd0b.jsonl
Checkpoint saved to /content/drive/MyDrive/checkpoint.pkl
```

## 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(0.6416), tensor(0.6214), tensor(0.4868), tensor(0.1001), tensor(0.2323), tensor(0.1683), tensor(0.7017), tensor(0.5877), tensor(
Average coherence score: 0.3794887363910675
```

Calcuate 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: 0.38372647762298584
Average coherence score for rejected suggestions: 0.36622926592826843
```