Entity Recognition report

Chi-Yeh Chen
chiyehc

Part1: Methodology

1. **Hypothesis**

- Prompt engineering: Providing both explicit instructions and a clear example. This is especially beneficial for improving model performance in Named Entity Recognition (NER) tasks, where consistent and precise output is crucial.
- Demonstration Selection: Selecting training examples based on semantic similarity to the test input (using embeddings) will improve the model's ability to generalize.

2. **Controlled Experiment Design**

- Prompt engineering:
  - Default natural language with inline tags (already in convert_bio_to_prompt) **(Baseline)**
  - Includes an in-context example, which is critical for few-shot learning. **(Experiment)**
- Demonstration Selection:
  - Random examples **(Baseline).**
  - Top-k similar examples based on cosine similarity (using Sentence Transformers embeddings). **(Experiment)**

3. **Experimental Setup**

- Modify get_chat_history:
  - Current Details:
    - Provide detailed instructions in the system_prompt to clarify the model's expected behavior (e.g., format of output, specific tags).
  - Expanded Details:
    - Explain the original version of get_chat_history and its shortcomings (e.g., random example selection, lack of context relevance).
    - A similarity-based selection mechanism using precomputed embeddings was added.
    - Incorporated structured prompt formats for better clarity.

4. **Steps to Implement**

- Baseline Setup (Baseline):
  - Describe the default setup:
    - Randomly selected 5-shot examples.
    - Simple narrative prompt format without specific formatting or examples.
    - Include the pseudocode for random example selection in `get_chat_history`.
- Add Similarity-Based Demonstration Selection (New):
  - Detail how the new approach was implemented:

- Computed embeddings for the training examples using `SentenceTransformer`.
- Retrieved the top 5 most similar examples for each input based on cosine similarity.

- Include the following key points:
  - Why cosine similarity was chosen (e.g., captures semantic relevance).
  - Example output of top n similar examples for an input text.

5. **Evaluation:**

- Run the dev set through all variations and compute F1 scores.
- Compare performance across experiments to identify significant patterns.
- Expected Results
  - Prompt Engineering and Demonstration Similarity: High-similarity examples are expected to yield better F1 scores than random or diverse selection because of better pattern alignment.

6. **Reference**

- In-Context Learning for Few-Shot Nested Named Entity Recognition
- PromptNER : Prompting For Named Entity Recognition
- Data augmentation via context similarity: An application to biomedical Named Entity Recognition

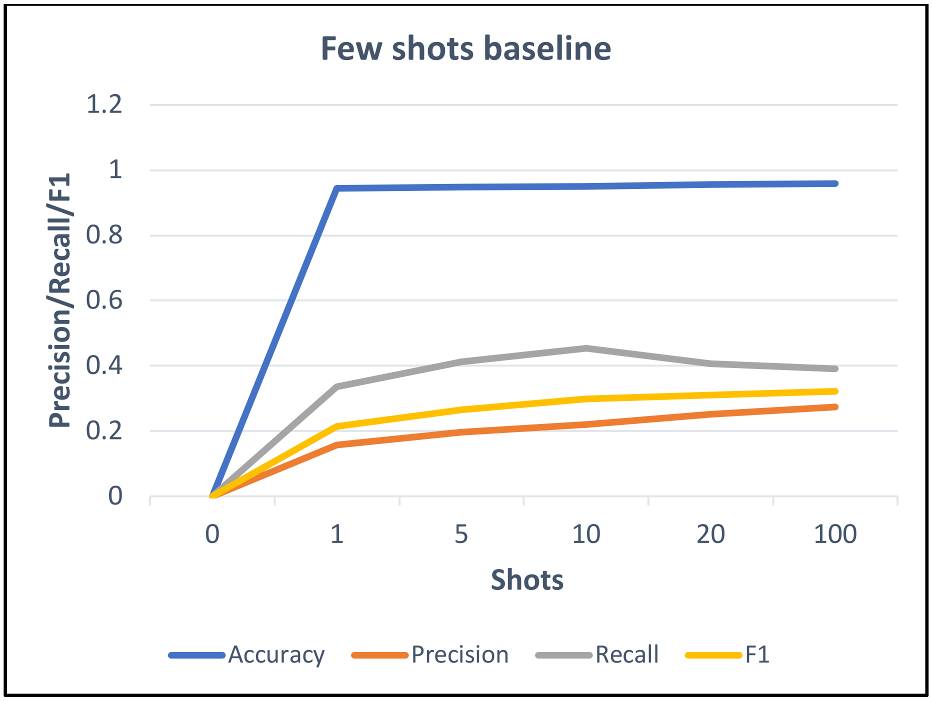
Part2: Experiment Results

Table 1: Baseline example results table.

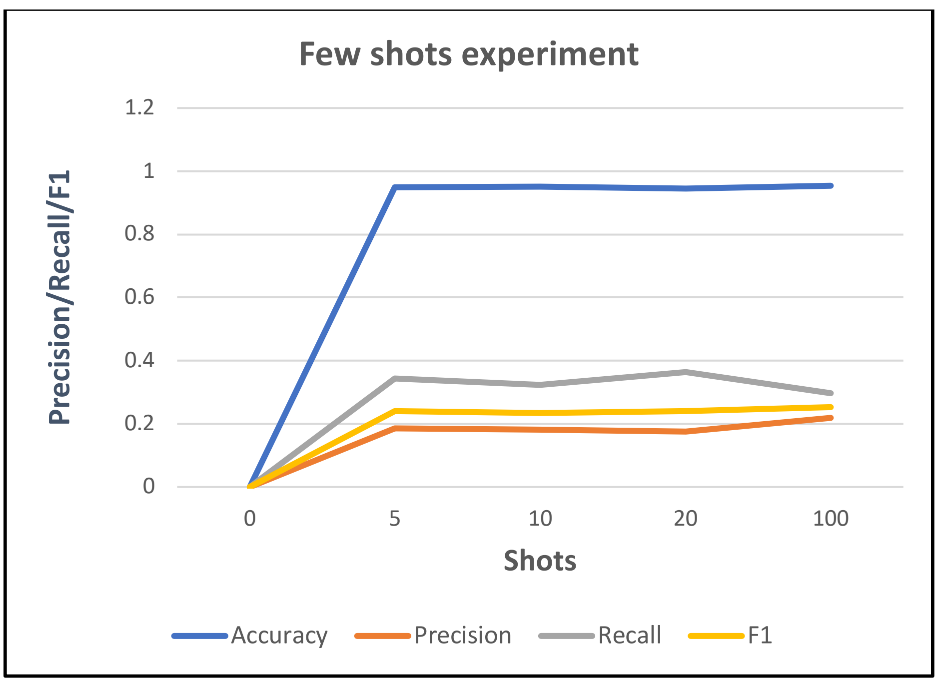
Approach	Shots	Accuracy	Precision	Recall	F1
Finetune BERT	**-**	**0.947**	**0.408**	**0.518**	**0.456**
Zero-shot LLM Baseline	0	0	0	0	0
Few-shot LLM Baseline	1	0.944	0.157	0.336	0.214
Few-shot LLM Baseline	5	0.948	0.196	0.412	0.266
Few-shot LLM Baseline	10	0.950	0.221	0.454	0.298
Few-shot LLM Baseline	20	0.956	0.252	0.406	0.311
Few-shot LLM Baseline	**100**	**0.959**	**0.274**	**0.391**	**0.322**

Table 2: Experiment example results table.

Approach	Shots	Accuracy	Precision	Recall	F1
Few-shot LLM Experiment	0	0	0	0	0
Few-shot LLM Experiment	5	0.95	0.185	0.343	0.24
Few-shot LLM Experiment	10	0.951	0.182	0.324	0.234
Few-shot LLM Experiment	20	0.945	0.176	0.364	0.24
Few-shot LLM Experiment	**100**	**0.954**	**0.219**	**0.297**	**0.253**



Plot 1. Few-shot examples of performance for the baseline model



Plot 2. Few-shot examples of performance for the experiment model

Part3: Analysis and Discussion

1. **Training settings:**
   - Model: gpt-4o-mini
   - Temperature: 0.5

2. **Analysis of Results**
   From the tables and figures, the following observations can be made:
   - Baseline Results:
     - The baseline model with few-shot examples shows a consistent improvement in **precision**, **recall**, and **F1-score** as the number of shots increases.
     - The best F1-score achieved by the baseline is **0.322** with **100 shots**, indicating that increasing the number of random examples helps the model understand the task better, even without similarity optimization.
   - Similarity-Based Example Selection:
     - The experiment using similarity-based selection demonstrates comparable performance but does not significantly outperform the baseline, which is a bit weird.
     - With 5, 10, 20, and 100 shots, the F1-scores remain largely similar to the baseline, suggesting that the similarity-based selection may not have leveraged additional context effectively.
     - In fact, there are slight performance dips in precision and recall compared to the baseline, which is counterintuitive since similarity-based selection is designed to provide more relevant examples.
   - Performance Trends:
     - Both approaches show that **accuracy** remains high and nearly identical across all shots, indicating the model consistently identifies the majority of tokens correctly (likely "O" tokens that are not part of entities).
     - **Precision** is consistently lower than recall, suggesting the model struggles with false positives (incorrectly identifying non-entities as entities).
     - The performance gains diminish as the number of shots increases (e.g., the difference between 20 and 100 shots is minimal), highlighting diminishing returns in adding more examples.

3. **Discussion of Similarity-Based Selection**
   The expectation with similarity-based example selection is that providing the model with more relevant, contextually similar examples would improve its ability to generalize. However, the results indicate otherwise. Possible reasons include:
   - Overfitting to Similarity:
     - Similar examples may lead the model to focus on highly specific patterns rather than generalizing across diverse contexts. This can result in lower precision or recall, particularly if the test example contains entities that deviate from the patterns seen in the selected examples.
   - Lack of Diversity:
     - Random selection may inherently introduce diverse examples, exposing the model to a wider variety of entity types and contexts. This diversity could be advantageous for generalization, whereas similarity-based selection could introduce redundancy.
   - Embedding Limitations:

    - The embeddings generated by the SentenceTransformer model may not perfectly capture the nuances of the task (e.g., BIO tagging and NER-specific relationships). The similarity metric may not align with the task's requirements.
- Prompt Design:
    - The system prompt and the way examples are presented may not sufficiently leverage the advantages of similarity-based selection. If the prompt does not highlight the relevance of examples, the model might not fully benefit from them.
- Evaluation Metrics:
    - High accuracy in both setups suggests the model performs well on "O" tokens but struggles with boundary detection and entity classification. This indicates that the evaluation metric might not fully capture the nuances of improvement for entities.

4. Proposed Improvements

- Enhance Embedding Selection:
    - Experiment with specific embeddings
    - Use supervised fine-tuning for embeddings if labeled data is available, aligning the similarity metric more closely with the task.
- Increase Diversity in Similar Examples:
    - Instead of selecting the top 5 most similar examples, combine similarity with diversity-based sampling (e.g., maximum marginal relevance) to ensure a broader coverage of entity types and contexts.
- Prompt Optimization:
    - Explicitly highlight the relevance of the selected examples in the system prompt.
    - Provide a structured explanation of why specific examples are relevant to the input.
- Error Analysis:
    - Perform a detailed analysis of errors (e.g., false positives, false negatives). Identify whether the issues are due to entity boundaries, entity types, or specific tokens.
- Weighted Example Selection:
    - Introduce a weighting mechanism that combines similarity scores with other criteria, such as entity diversity or example complexity.

5. Conclusion

- Key Findings:
    - Similarity-based selection performs comparably to random selection but does not significantly improve performance.
    - The diminishing returns of adding more examples highlight the need for optimizing example selection strategies beyond simple similarity metrics.
- Future Directions:
    - Explore hybrid selection methods that balance similarity and diversity.
    - Refine the prompt design to better utilize in-context examples.
    - Investigate advanced evaluation methods to capture task-specific improvements.

Appendix:

- <u>Baseline</u>

HW4: LLM prompting for entity labeling

This notebook contains starter code for prompting an LLM API for the task of entity recognition. It has minimal text so you can easily copy it to **handin.py** when you submit. Please read all the comments in the code as they contain important information.

```python
# This code block just contains standard setup code for running in Python
import json
import string
import re
import time
from tqdm.auto import tqdm

# PyTorch imports
import torch
from torch.utils.data import DataLoader
import numpy as np

# Fix the random seed(s) for reproducability
random_seed = 8942764
torch.random.manual_seed(random_seed)
torch.cuda.manual_seed(random_seed)
np.random.seed(random_seed)

#!pip install ipytest
#!pip install transformers
#!pip install datasets
#!pip install evaluate
#!pip install seqeval
#!pip install ratelimit

from transformers import AutoTokenizer, BertModel, DefaultDataCollator

from datasets import load_dataset

import evaluate
from ratelimit import limits

# Just a helper function for efficiently removing punctuation from a string
def strip_punct(s):  return s.translate(string.punctuation)
```

[2]
```python
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).
```

Prepare data | Processing

```python
# Load the dataset
from datasets import Dataset, ClassLabel, Sequence

data_splits = load_dataset('json', data_files={'train': '/content/drive/MyDrive/Colab Notebooks/HW4/dinos_and_deities_train_bio.jsonl',

# Load dicts for mapping int labels to strings, and vice versa
label_names_fname = "/content/drive/MyDrive/Colab Notebooks/HW4/dinos_and_deities_train_bio.jsonl.labels"
labels_int2str = []
with open(label_names_fname) as f:
    labels_int2str = f.read().split()
print(f"Labels: {labels_int2str}")
labels_str2int = {l: i for i, l in enumerate(labels_int2str)}

# Also create a set containing the original labels, without B- and I- tags
orig_labels = set()
for label in labels_str2int.keys():
    orig_label = label[2:]
    if orig_label:
        orig_labels.add(orig_label)
print(f"Orig labels: {orig_labels}")

# data_splits.cast_column("ner_tags", Sequence(ClassLabel(names=labels_int2str)))
print(data_splits)
```

Generating train split: 1749/0 [00:00<00:00, 2592.25 examples/s]

Generating dev split: 150/0 [00:00<00:00, 445.78 examples/s]

Generating test split: 303/0 [00:00<00:00, 642.90 examples/s]

```
Labels: ['I-Aquatic_animal', 'B-Deity', 'B-Mythological_king', 'I-Mythological_king', 'I-Cretaceous_dinosaur', 'B-Aquatic_animal', 'B-A
Orig labels: {'Aquatic_animal', 'Goddess', 'Deity', 'Cretaceous_dinosaur', 'Mythological_king', 'Aquatic_mammal'}
DatasetDict({
    train: Dataset({
        features: ['para_index', 'title', 'doc_id', 'content', 'page_id', 'id', 'tokens', 'ner_strings', 'ner_tags'],
        num_rows: 1749
    })
    dev: Dataset({
        features: ['para_index', 'title', 'doc_id', 'content', 'page_id', 'id', 'tokens', 'ner_strings', 'ner_tags'],
        num_rows: 150
    })
    test: Dataset({
        features: ['para_index', 'title', 'doc_id', 'content', 'page_id', 'id', 'tokens', 'ner_strings', 'ner_tags'],
        num_rows: 303
    })
})
```

OpenAI version

Prepare Chat Messages | Send a Request

```
!pip install openai --force-reinstall -v "openai==1.55.3"
```

顯示隱藏的輸出內容

```
from openai import OpenAI

# Use the API key that we
client = OpenAI(api_key='sk-proj-Bv99Y-6Zgl0JUdT9UOEsj3chszx3es_RxXhYD2L1NtQXFMN9aWSxpwbgNFFgniK-BBIUXwEPnLT3BlbkFJdnjoJUqZoBEXP5GjkmGjZFtoEgRjcjS5awUrrbYNeO_

USER_STR = "user"
SYSTEM_STR = "system"
MSG_STR = "content"
```

[4]
```
# Here is how you can use the API to prompt the OpenAI model.
# Docs: https://platform.openai.com/docs/api-reference
messages = [
    {'role': SYSTEM_STR, MSG_STR:
     """You will be given input text containing different types of entities that you will label.
     This is the list of entity types to label: Deity, Mythological_king, Cretaceous_dinosaur, Aquatic_mammal, Aquatic_animal, Goddess.
     Label the enities by surrounding them with tags like '<Cretaceous_dinosaur> Beipiaognathus </Cretaceous_dinosaur>'."""
     },
     {'role': USER_STR, MSG_STR: """Text: Once paired in later myths with her Titan brother Hyperion as her husband, mild-eyed Euryphaessa, the far-shining on
     {'role': SYSTEM_STR, MSG_STR: """Labels: Once paired in later myths with her Titan brother <Deity> Hyperion </Deity> as her husband, mild-eyed Euryphaess
     {'role': USER_STR, MSG_STR: """Text: From her ideological conception, Taweret was closely grouped with (and is often indistinguishable from) several othe
]

# # This is where you provide the final prompt that we want the model to complete to give us the answer.
# message = f"""Text: From her ideological conception, Taweret was closely grouped with (and is often indistinguishable from) several other protective hippopo
# Labels: """

response = client.chat.completions.create(
    model="gpt-4o-mini",
    temperature=0.5,
    seed=random_seed,
    messages=messages
)

print(response.choices[0].message.content)

# You can also print out the usage, in number of tokens.
# Pricing is per input/output token, listed here: https://openai.com/pricing
print(f"Usage: {response.usage.prompt_tokens} input, {response.usage.completion_tokens} output, {response.usage.total_tokens} total tokens")
```

```
From her ideological conception, <Goddess> Taweret </Goddess> was closely grouped with (and is often indistinguishable from) several other protective <Goddess
Usage: 307 input, 87 output, 394 total tokens
```

[33]
```
# Ok, now let's make the prompting a bit more programmatic. First, implement a function that takes an example from
# the dataset, and converts it into a message for the model using the format we specified above.
# You might want to use the Python string "format" function to make this a bit easier, especially since
# You will be experimenting with different prompts later.
#
# TODO: implement this.
def get_message(example):
    """
    Converts an example into a single user message for the model.

    Args:
        example (dict): A single example from the dataset.
                        Expected keys: 'tokens' (list of words).

    Returns:
        str: The user message content as a string.
    """
    # Retrieve the text content by joining the tokens
    text = " ".join(example["tokens"])  # Combine tokens into a single string

    # Format the text for the user message
    user_message_content = text

    return user_message_content
```

```python
# Next we're going to implement a function to return the chat_history, but in order to do that we first need
# to be able to convert labeled examples from the dataset into a format that makes more sense for the model,
# in this case the HTML-style format we specified in the example. That's the task for this function: take
# an example from the dataset as input, and return a string that has tagged the text with labels in the given
# HTML-style format.
#
# TODO: implement this.
def convert_bio_to_prompt(example):
    """
    Converts a labeled example from the dataset into an HTML-style formatted string
    with entities tagged according to the specified BIO labels.

    Args:
        example (dict): A single example from the dataset.
                        Expected keys: 'tokens' (list of words) and 'ner_strings' (list of BIO labels).

    Returns:
        str: A string where entities in the text are tagged with the specified HTML-style format.
    """
    tokens = example["tokens"]  # List of tokens
    ner_strings = example["ner_strings"]  # Corresponding BIO labels

    # Initialize variables for building the output string
    formatted_text = ""
    current_entity = None
    current_entity_tokens = []

    # Iterate over tokens and their corresponding BIO tags
    for token, s in zip(tokens, ner_strings):
        if s == "O":  # If the token is outside any entity
            if current_entity:  # Close the current entity tag
                formatted_text += f"<{current_entity[2:]}> {' '.join(current_entity_tokens)} </{current_entity[2:]}> "
                current_entity = None
                current_entity_tokens = []
            formatted_text += token + " "  # Add the token as normal text
        else:
            # Use the label directly if it's not 0
            if current_entity == s:  # Continue the current entity
                current_entity_tokens.append(token)
            else:
                if current_entity:  # Close the previous entity tag
                    formatted_text += f"<{current_entity[2:]}> {' '.join(current_entity_tokens)} </{current_entity[2:]}> "
                # Start a new entity
                current_entity = s
                current_entity_tokens = [token]

    # Handle the last entity if it exists
    #if current_entity:
    #    formatted_text += f"<{current_entity}> {' '.join(current_entity_tokens)} </{current_entity}> "

    return formatted_text.strip()  # Return the formatted text, removing trailing spaces.
```

```python
# Now we can write a function that takes the number of shots, dataset, list of entity types, and
# convert_bio_to_prompt function, and returns the chat_history (a list of maps) structured as in
# the example.
#
# TODO: implement this.
def get_chat_history(shots, dataset, entity_types_list, convert_bio_to_prompt_fn):
    """
    Generates a chat history formatted as a list of maps for few-shot learning.

    Args:
        shots (int): Number of examples to include in the chat history (few-shot examples).
        dataset (list): The dataset containing examples (list of dictionaries with 'tokens' and 'ner_tags').
        entity_types_list (list): List of entity types to include in the system prompt.
        convert_bio_to_prompt_fn (function): Function that converts labeled examples to the desired prompt format.

    Returns:
        list: Chat history structured as a list of dictionaries with roles and content.
    """
    # Create the system prompt
    system_prompt = {
        "role": "system",
        "content": (
            f"You will be given input text containing different types of entities that you will label.\n"
            f"This is the list of entity types to label: {', '.join(entity_types_list)}.\n"
            f"Label the entities by surrounding them with tags like '<Entity_Type> Entity </Entity_Type>'."
        )
    }

    # Initialize the chat history with the system prompt
    chat_history = [system_prompt]

    # Add the specified number of examples (shots) from the dataset
    for i in range(min(shots, len(dataset))):
        example = dataset[i]

        # Convert the example to the prompt format
        formatted_example = convert_bio_to_prompt_fn(example)
        #print(formatted_example)

        # Add the user message (text input)
        user_message = {
            "role": "user",
            "content": f"{' '.join(example['tokens'])}"
        }
        chat_history.append(user_message)

        # Add the assistant message (labeled output)
        assistant_message = {
            "role": "system",
            "content": f"{formatted_example}"
        }
        chat_history.append(assistant_message)

    return chat_history
```

[37]

```python
### OpenAI
# Now let's wrap that call in a function that takes shots and an example, calls the API and returns the response.
def call_api_openai(shots, example):
    success = False
    #print(type(example['tokens']), example['tokens'])

    while not success:
        try:
            chat_history = get_chat_history(shots, data_splits['train'], orig_labels, convert_bio_to_prompt)
            message = {'role': USER_STR, 'content': get_message(example)}
            chat_history.append(message)
            response = client.chat.completions.create(
                model="gpt-4o-mini",
                temperature=0.5,
                messages=chat_history
            )
            success = 1
        except Exception as err:
            tqdm.write(f"Caught exception: {err}")
    return response.choices[0].message.content
```

```python
# Now we want to be able to evaluate the model, in order to compare it to e.g. the fine-tuned BERT model.
# In order to do this, we need to write the reverse of the convert_bio_to_prompt function, so that we can
# convert in the other direction, from the generated response in prompt format, back to bio for evaluation
# using seqeval.
#
# The input to this function is the string response from the model, and the output should be a list of
# text BIO labels corresponding to the labeling implied by the tagged output produced by the model, as
# well as the list of tokens (since the generative model could return something different than we gave it,
# and we need to handle that somehow in the eval).
#
# TODO: implement this
import re
import string

def convert_response_to_bio(response):
    """
    Converts the model-generated response with HTML-style tags back into BIO format.

    Args:
        response (str): The generated response from the model in HTML-style format.

    Returns:
        tuple: A tuple containing:
            - tokens (list of str): The tokens extracted from the response.
            - bio_labels (list of str): The corresponding BIO labels for the tokens.
    """
        # Remove the 'Labels:' prefix if it exists
    if response.startswith('Labels:'):
        response = response[len('Labels:'):].strip()

    tokens = []
    bio_labels = []

    # Regular expression to match tags and plain text
    tag_pattern = re.compile(r"(</?[\w\-]+>)|([^<>]+)")  # Matches <tag>, </tag>, and plain text

    current_label = "O"  # Start with "O" (outside any entity)
    inside_entity = False  # Track whether we are inside an entity tag

    for match in tag_pattern.finditer(response):
        tag_or_text = match.group()

        if tag_or_text.startswith("</"):  # Closing tag
            current_label = "O"
            inside_entity = False
        elif tag_or_text.startswith("<"):  # Opening tag
            current_label = tag_or_text[1:-1]  # Extract tag name without <>
            inside_entity = True
        else:
            # Process plain text
            for i, token in enumerate(tag_or_text.split()):
                tokens.append(token)

                if inside_entity:
                    #bio_labels.append(current_label)
                    if i == 0:
                        bio_labels.append(f"B-{current_label}")  # Start of an entity
                    else:
                        bio_labels.append(f"I-{current_label}")  # Continuation of the same entity
                else:
                    bio_labels.append("O")  # Outside any entity

    punctuations = set(string.punctuation)
    merged_tokens = []
    merged_bio_labels = []

    for token, label in zip(tokens, bio_labels):
        if token in punctuations and merged_tokens:
            merged_tokens[-1] += token
        else:
            merged_tokens.append(token)
            merged_bio_labels.append(label)

    return merged_bio_labels, merged_tokens
```

```
html_str = 'From <Goddess> her</Goddess> ideological conception, <Goddess> the deity Taweret </Goddess> was closely grouped with (and is often indistinguishable
labels, text = convert_response_to_bio(html_str)
true_labels = ['O', 'B-Goddess', 'O', 'O', 'B-Goddess', 'I-Goddess', 'I-Goddess', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-Aquatic_mammal',
true_text = ['From', 'her', 'ideological', 'conception,', 'the', 'deity', 'Taweret', 'was', 'closely', 'grouped', 'with', '(and', 'is', 'often', 'indistinguishab
print(labels)
print(text)
assert len(labels) == len(true_labels)
assert len(text) == len(true_text)
```

```
['O', 'B-Goddess', 'O', 'O', 'B-Goddess', 'I-Goddess', 'I-Goddess', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-Aquatic_mammal', 'B-Goddess',
['From', 'her', 'ideological', 'conception,', 'the', 'deity', 'Taweret', 'was', 'closely', 'grouped', 'with', '(and', 'is', 'often', 'indistinguishable', 'from)'
```

```
# Here's a test example you can use to validate/debug your code (note that this was constructed to simulate various
# spacing/tokenization scenarios and does not necessarily reflect "correct" labeling wrt the training data):
import ipytest
ipytest.autoconfig()
def test_convert_html_to_bio():
    html_str = 'From <Goddess> her</Goddess> ideological conception, <Goddess> the deity Taweret </Goddess> was closely grouped with (and is often indistinguisha
    labels, text = convert_response_to_bio(html_str)
    true_labels = ['O', 'B-Goddess', 'O', 'O', 'B-Goddess', 'I-Goddess', 'I-Goddess', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-Aquatic_mamm
    true_text = ['From', 'her', 'ideological', 'conception,', 'the', 'deity', 'Taweret', 'was', 'closely', 'grouped', 'with', '(and', 'is', 'often', 'indistingui
    print(labels)
    print(text)
    assert labels == true_labels
    assert text == true_text

def test_convert_html_to_bio_labels():
    html_str = 'Labels: From <Goddess> her</Goddess> ideological conception, <Goddess> the deity Taweret </Goddess> was closely grouped with (and is often indist
    labels, text = convert_response_to_bio(html_str)
    true_labels = ['O', 'B-Goddess', 'O', 'O', 'B-Goddess', 'I-Goddess', 'I-Goddess', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-Aquatic_mamm
    true_text = ['From', 'her', 'ideological', 'conception,', 'the', 'deity', 'Taweret', 'was', 'closely', 'grouped', 'with', '(and', 'is', 'often', 'indistingui
    print(labels)
    print(text)
    assert labels == true_labels
    assert text == true_text

ipytest.run('-vv')  # '-vv' for increased verbosity
```

```
======================================= test session starts ========================================
platform linux -- Python 3.10.12, pytest-8.3.4, pluggy-1.5.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.7.0, typeguard-4.4.1
collecting ... collected 2 items

t_9ab71cf2779f41a5858c4460f4aaf63a.py::test_convert_html_to_bio PASSED                  [ 50%]
t_9ab71cf2779f41a5858c4460f4aaf63a.py::test_convert_html_to_bio_labels PASSED           [100%]

========================================= warnings summary =========================================
../usr/local/lib/python3.10/dist-packages/_pytest/config/__init__.py:1277
  /usr/local/lib/python3.10/dist-packages/_pytest/config/__init__.py:1277: PytestAssertRewriteWarning: Module already imported so cannot be rewritten: anyio
    self._mark_plugins_for_rewrite(hook)

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
=================================== 2 passed, 1 warning in 0.07s ===================================
<ExitCode.OK: 0>
```

[22]
```
# Now we can put all of the above together to evaluate!
metric = evaluate.load("seqeval")
output_path = "test_predictions_llm.json"
def run_eval(dataset, shots):
  all_predictions = []

  for example in tqdm(dataset, total=len(dataset), desc="Evaluating", position=tqdm._get_free_pos()):

      # String list of labels (BIO)
      true_labels = [labels_int2str[l] for l in example['ner_tags']]
      example["tokens"] = [t if isinstance(t, str) else " ".join(t) for t in example["tokens"]]

      example_tokens = example['tokens']

      response_text = call_api_openai(shots, example)
      #print(f'response text: { response_text}')

      # String list of predicted labels (BIO)
      predictions, generated_tokens = convert_response_to_bio(response_text)
      all_predictions.append(predictions)

      # Handle case where the generated text doesn't align with the input text.
      # Basically, we'll eval everything up to where the two strings start to diverge.
      # We relax this slightly by ignoring punctuation (sometimes we lose a paren or something,
      # but that's not catastrophic for eval/tokenization).
      # Just predict 'O' for anything following mismatch.
      matching_elements = [strip_punct(i) == strip_punct(j) for i, j in zip(example_tokens, generated_tokens)]

      if False in matching_elements:
         last_matching_idx = matching_elements.index(False)
      else:
         last_matching_idx = min(len(generated_tokens), len(example_tokens))

      predictions = predictions[:last_matching_idx] + ['O']*(len(example_tokens)-last_matching_idx)
      metric.add(predictions=predictions, references=true_labels)

  return metric.compute(zero_division=0)
```

```
# Run the eval on the dev set
dev_examples_to_take = 0

dev_set = data_splits['dev']
if dev_examples_to_take > 0:
    dev_set = data_splits['dev'].select(range(dev_examples_to_take))

for num_shots in [0, 1, 5, 10, 20, 100]:  # Test with different numbers of examples
    print(f"shots: {num_shots}")
    result = run_eval(dev_set, shots=num_shots)
    print(f"Results for {num_shots} shots: {result}")
```

```
shots: 0
Evaluating: 100% 150/150 [04:20<00:00,  1.55s/it]
Results for 0 shots: {'Aquatic_animal': {'precision': 0.0, 'recall': 0.0, 'f1': 0.0, 'number': 62}, 'Aquatic_mammal': {'precision': 0.0, 'recall': 0.0, 'f1': 0.0, 'number': 35}, 'Cretaceous_dinosaur': {'
shots: 1
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Goddess seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Deity seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Mythological_king seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Cretaceous_dinosaur seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Aquatic_mammal seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Aquatic_animal seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Tamayo seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Diety seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
Evaluating: 100% 150/150 [05:34<00:00,  4.11s/it]
Results for 1 shots: {'Aquatic_animal': {'precision': 0.043795620437956206, 'recall': 0.0967741935483871, 'f1': 0.06030150753768844, 'number': 62}, 'Aquatic_mammal': {'precision': 0.0989010989010989, 're
shots: 5
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Mammal seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
Evaluating: 100% 150/150 [07:02<00:00,  4.09s/it]
Results for 5 shots: {'Aquatic_animal': {'precision': 0.06382978723404255, 'recall': 0.0967741935483871, 'f1': 0.07692307692307691, 'number': 62}, 'Aquatic_mammal': {'precision': 0.15384615384615385, 're
shots: 10
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: H-BG_Mythological_king seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: H-BG_Deity seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: Nipponosaurus seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
Evaluating: 100% 150/150 [06:05<00:00,  3.48s/it]
Results for 10 shots: {'Aquatic_animal': {'precision': 0.1111111111111111, 'recall': 0.1935483870967742, 'f1': 0.1411764705882353, 'number': 62}, 'Aquatic_mammal': {'precision': 0.13924050632911392, 'rec
shots: 20
Evaluating: 100% 150/150 [05:42<00:00,  3.37s/it]
Results for 20 shots: {'Aquatic_animal': {'precision': 0.11702127659574468, 'recall': 0.1774193548387097, 'f1': 0.14102564102564102, 'number': 62}, 'Aquatic_mammal': {'precision': 0.12727272727272726, 'r
shots: 100
Evaluating: 100% 150/150 [22:45<00:00, 10.04s/it]
Results for 100 shots: {'Aquatic_animal': {'precision': 0.136986301369863, 'recall': 0.16129032258064516, 'f1': 0.14814814814814814, 'number': 62}, 'Aquatic_mammal': {'precision': 0.09333333333333334, 'r
/usr/local/lib/python3.10/dist-packages/seqeval/metrics/sequence_labeling.py:171: UserWarning: C-Aquatic_animal seems not to be NE tag.
  warnings.warn('{} seems not to be NE tag.'.format(chunk))
```

Test json

```python
def run_test(dataset, shots, output_filename="test_predictions_llm_baseline.json"):

    all_predictions = []

    for example in tqdm(dataset, total=len(dataset), desc="Evaluating", position=tqdm._get_free_pos()):
        # String list of labels (BIO)
        #true_labels = [labels_int2str[l] for l in example['ner_tags']]
        example_tokens = example['tokens']

        response_text = call_api_openai(shots, example)

        # String list of predicted labels (BIO)
        predictions, generated_tokens = convert_response_to_bio(response_text)

        # Handle case where the generated text doesn't align with the input text
        matching_elements = [strip_punct(i) == strip_punct(j) for i, j in zip(example_tokens, generated_tokens)]

        if False in matching_elements:
            last_matching_idx = matching_elements.index(False)
        else:
            last_matching_idx = min(len(generated_tokens), len(example_tokens))

        # Adjust predictions for mismatch
        predictions = predictions[:last_matching_idx] + ['O'] * (len(example_tokens) - last_matching_idx)
        print(predictions)
        # Save predictions for this sentence
        all_predictions.append(predictions)

    # Write predictions to the JSON file
    with open(output_filename, "w") as f:
        json.dump(all_predictions, f, indent=4)

    print(f"Predictions saved to {output_filename}")
```

```python
# Load dicts for mapping int labels to strings, and vice versa
test_data_path = "/content/drive/MyDrive/Colab Notebooks/HW4/dinos_and_deities_test_bio_nolabels.jsonl"
with open(test_data_path, "r") as f:
    test_data = [json.loads(line.strip()) for line in f]

# Convert test data into a Hugging Face Dataset
test_data = Dataset.from_list(test_data)
print(test_data)

label_names_fname = "/content/drive/MyDrive/Colab Notebooks/HW4/dinos_and_deities_train_bio.jsonl.labels"
labels_int2str = []
with open(label_names_fname) as f:
    labels_int2str = f.read().split()
print(f"Labels: {labels_int2str}")
labels_str2int = {l: i for i, l in enumerate(labels_int2str)}

# Also create a set containing the original labels, without B- and I- tags
orig_labels = set()
for label in labels_str2int.keys():
    orig_label = label[2:]
    if orig_label:
        orig_labels.add(orig_label)
print(f"Orig labels: {orig_labels}")

print(f"Labels in label file: {labels_int2str}")
print(f"Original labels detected: {orig_labels}")
```

```
Dataset({
    features: ['para_index', 'title', 'doc_id', 'content', 'page_id', 'id', 'tokens', 'ner_strings', 'ner_tags'],
    num_rows: 303
})
Labels: ['I-Aquatic_animal', 'B-Deity', 'B-Mythological_king', 'I-Mythological_king', 'I-Cretaceous_dinosaur', 'B-Aquatic_animal', 'B-Aquatic_mammal', '
Orig labels: {'Cretaceous_dinosaur', 'Mythological_king', 'Goddess', 'Deity', 'Aquatic_mammal', 'Aquatic_animal'}
Labels in label file: ['I-Aquatic_animal', 'B-Deity', 'B-Mythological_king', 'I-Mythological_king', 'I-Cretaceous_dinosaur', 'B-Aquatic_animal', 'B-Aqua
Original labels detected: {'Cretaceous_dinosaur', 'Mythological_king', 'Goddess', 'Deity', 'Aquatic_mammal', 'Aquatic_animal'}
```

```python
[20] for num_shots in [10]:  # Test with different numbers of examples
    print(f"shots: {num_shots}")
    result = run_test(test_data, shots=num_shots)
    print(f"Results for {num_shots} shots: {result}")
```

```
shots: 10
Evaluating: 100% 303/303 [12:20<00:00,  2.00s/it]
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
['B-Aquatic_animal', 'B-Aquatic_animal', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
['B-Kami', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
['B-Aquatic_animal', 'B-Aquatic_animal', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', '
```

- Experiment

Prompt Formatting | Demonstration selection

```python
from sentence_transformers import SentenceTransformer, util

# Initialize the embedding model (you can load it once globally)
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

# Precompute embeddings for the training dataset
def compute_train_embeddings(dataset):
    """
    Compute embeddings for the training dataset.

    Args:
        dataset (list): The training dataset, where each example is a dictionary with 'tokens'.

    Returns:
        list: A list of embeddings for the training examples.
    """
    texts = [" ".join(example['tokens']) for example in dataset]
    embeddings = embedding_model.encode(texts, convert_to_tensor=True)
    return embeddings

# Assume `train_embeddings` is precomputed
train_embeddings = compute_train_embeddings(data_splits['train'])

print(train_embeddings)
```

```
/usr/local/lib/python3.10/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 in your Google
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(
modules.json: 100% 349/349 [00:00<00:00, 14.6kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 5.50kB/s]
README.md: 100% 10.7k/10.7k [00:00<00:00, 359kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 2.35kB/s]
config.json: 100% 612/612 [00:00<00:00, 27.1kB/s]
model.safetensors: 100% 90.9M/90.9M [00:00<00:00, 190MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 18.1kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 2.73MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 5.02MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 5.62kB/s]
1_Pooling/config.json: 100% 190/190 [00:00<00:00, 12.1kB/s]
tensor([[-0.0425,  0.0209, -0.0332,  ..., -0.0151,  0.0373, -0.0159],
        [-0.0393, -0.0044, -0.0260,  ..., -0.1188, -0.0044,  0.0230],
        [-0.0357,  0.0892,  0.0373,  ..., -0.0463, -0.0062,  0.0024],
        ...,
        [ 0.0475,  0.0120, -0.0962,  ...,  0.0660,  0.0012, -0.0173],
        [ 0.0299,  0.0234,  0.0084,  ...,  0.0572, -0.0026,  0.0155],
        [-0.0044,  0.0668,  0.0368,  ...,  0.0075, -0.0335,  0.0644]])
```

```python
def get_similar_examples(input_text, train_data, embeddings, top_k=None):
    """
    Retrieve all examples from the training dataset sorted by cosine similarity with the input text.

    Args:
        input_text (str): The input text for which we want similar examples.
        train_data (list): The training dataset.
        embeddings (torch.Tensor): Precomputed embeddings for the training data.
        top_k (int, optional): Number of similar examples to retrieve. If None, return all sorted examples.

    Returns:
        list: Sorted examples from the training data based on cosine similarity.
    """
    input_embedding = embedding_model.encode(input_text, convert_to_tensor=True)
    cosine_scores = util.cos_sim(input_embedding, embeddings).squeeze(0)

    # Get the sorted indices based on cosine similarity
    sorted_indices = torch.argsort(cosine_scores, descending=True).tolist()

    # Return all examples sorted by cosine similarity, or top_k if specified
    if top_k:
        sorted_indices = sorted_indices[:top_k]

    return [train_data[i] for i in sorted_indices]
```

```python
def get_chat_history(shots, dataset, entity_types_list, convert_bio_to_prompt_fn):
    """
    Generates a chat history formatted as a list of maps for few-shot learning.

    Args:
        shots (int): Number of examples to include in the chat history (few-shot examples).
        dataset (list): The dataset containing examples (list of dictionaries with 'tokens' and 'ner_tags').
        entity_types_list (list): List of entity types to include in the system prompt.
        convert_bio_to_prompt_fn (function): Function that converts labeled examples to the desired prompt format.

    Returns:
        list: Chat history structured as a list of dictionaries with roles and content.
    """
    system_prompt = {
        "role": "system",
        "content": (
            f"You are a highly capable NER labeling model. Your task is to extract and tag entities in the input "
            f"text according to the BIO format. Entity types include: {', '.join(entity_types_list)}.\n\n"
            f"Example of tagging:\n"
            f"Input: 'John Doe works at Acme Corp in New York.'\n"
            f"Output: 'John Doe <Person> works at <Organization> Acme Corp </Organization> in New York <Location>'.\n\n"
            f"Format your output with the tags exactly as shown. Use 'O' for words that do not belong to an entity."
        )
    }
    # Initialize the chat history with the system prompt
    chat_history = [system_prompt]

    # Input text for similarity-based selection (example: from the dev/test dataset)
    input_text = " ".join(dataset[0]['tokens'])  # Replace dataset[0] with the actual input example
#
    ## Select the top-k similar examples
    similar_examples = get_similar_examples(input_text, data_splits['train'], train_embeddings)

    # Add the selected examples to the chat history
    for i in range(min(shots, len(similar_examples))):

        # Convert the example to the prompt format
        example = similar_examples[i]
        #print(example)
        formatted_example = convert_bio_to_prompt_fn(example)

        # Add the user message (text input)
        user_message = {
            "role": "user",
            "content": f"{' '.join(example['tokens'])}"
        }
        #print(type(example['tokens']), example['tokens'])

        chat_history.append(user_message)

        # Add the assistant message (labeled output)
        assistant_message = {
            "role": "system",
            "content": f"{formatted_example}"
        }
        chat_history.append(assistant_message)

    return chat_history
```

[28]
```python
# Precompute training embeddings
#train_embeddings = compute_train_embeddings(data_splits['train'])

def call_api_openai(shots, example):
    success = False
    #print(type(example['tokens']), example['tokens'])

    while not success:
        try:
            # Retrieve tokens from the current example to compute similarity
            #input_text = " ".join(flatten_list(example["tokens"]))
            #print(input_text)
            # Get top-k similar examples
            #similar_examples = get_similar_examples(
            #    input_text, data_splits['train'], train_embeddings, top_k=shots
            #)
            #print(similar_examples)
            # Generate chat history using similar examples
            chat_history = get_chat_history(
                shots, data_splits['train'], orig_labels, convert_bio_to_prompt
            )

            # Add the message for the current example
            message = {'role': USER_STR, 'content': get_message(example)}
            chat_history.append(message)

            # Call the OpenAI API
            response = client.chat.completions.create(
                model="gpt-4o-mini",
                temperature=0.5,
                messages=chat_history
            )
            success = 1
        except Exception as err:
            tqdm.write(f"Caught exception: {err}")
    return response.choices[0].message.content
```

```python
# Run the evaluation
dev_set = data_splits['dev']  # Development set

for num_shots in [5, 10, 20, 100]:  # Test with different numbers of examples
    print(f"shots: {num_shots}")
    result = run_eval(dev_set, shots=num_shots)
    print(f"Results for {num_shots} shots: {result}")
```

```
shots: 5
Evaluating: 100% 150/150 [08:36<00:00,  4.66s/it]
Results for 5 shots: {'Aquatic_animal': {'precision': 0.046511627906976744, 'recall': 0.06451612903225806, 'f1': 0.05405405
shots: 10
```