ablation-experimentcode

November 29, 2023

```
[ ]: !zip -r prompts_v2.zip experiment_prompts
```

1 Ablation Experiments List:

1. Bilingual Prompting, but randomize sequence of translation given in prompt.*
2. Bilingual Prompting, but remove some of the most common/rare words translation, and see how much it hurts the model performance.
3. Fewshot Prompting, but switch location of your data. Instead of fewshot first, dictionary entries first
4. Fewshot Prompting, but only give 5 random examples
5. Fewshot Prompting, but, change a couple of target word from fewshot examples to become a dictionary entries instead.
6. Kitchen Sink: Gives full bilingual dictionary, then fewshot examples.

*Currently, the sequence of translation is almost ideal. Ideal sequence of translation can only be achieved if we repeat word level translation sequences.

```
[ ]: !wget https://artificial-language-prerequisites.s3.amazonaws.com/
     ↪Experiment_Data.zip
```

```
[ ]: import zipfile

     ZIP_FILE_PATH = "Experiment_Data.zip"
     TEMP_DIR = "tempextract/"
     EXTRACT_DIR = "extract/"

     import os
     os.makedirs(TEMP_DIR, exist_ok=True)
     with zipfile.ZipFile(ZIP_FILE_PATH, 'r') as zip_ref:
       zip_ref.extractall(TEMP_DIR)

     import shutil
     for root, dirs, files in os.walk(TEMP_DIR):
       for dir in dirs:
         os.makedirs(os.path.join(EXTRACT_DIR, dir), exist_ok=True)
       for file_ in files:
         group = file_.split('_')[0]
```

```
    target_dir = os.path.join(EXTRACT_DIR, group)

    source_file = os.path.join(root, file_)
    if os.path.exists(target_dir) and os.path.isdir(target_dir):
      shutil.copy2(source_file, target_dir)
    else:
      shutil.copy2(source_file, EXTRACT_DIR)

import shutil
shutil.rmtree(TEMP_DIR)
```

[ ]:
```
## Helper func
def load_map(f_name):
  import pickle
  with open(f_name, 'rb') as f:
    return pickle.load(f)

def load_file(f_name):
  with open(f_name, 'r') as f:
    return f.readlines()
## End of helper func
```

2 Prompts v1

2.1 1. Bilingual Prompting with randomized sequence

2.1.1 AB

[ ]:
```
import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vso')
]

NOISING_MAP = load_map(os.path.join(WORK_DIR, 'AB/AB_noising_map.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'B/B_compound_map.pickle'))

EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

[ ]:
```
def tokenize(text):
  import re
  import string
  ret = []
  for token in text.split(' '):
    result_list = re.findall(r'\w+|[^\w\s]', token)
    flag = 0
    for token in result_list:
      if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

import random
random.seed(2023)
def AB_bilingual_prompting(input_sentence, label_sentence, noising_map,␣
 ↪compound_map, word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_AB_bilingual_prompting
  # Uses the tokenize function above
  ### End of Explanation
  LABEL_TOKENS = tokenize(label_sentence)
  R_NOISING_MAP = {}
  for k,v in noising_map.items():
    R_NOISING_MAP[v] = k

  R_COMPOUND_MAP = {}
  for k,v in compound_map.items():
    R_COMPOUND_MAP[v] = k
```

```
NOISED_TOKENS = []
COMPOUNDED_TOKENS = []

for token in LABEL_TOKENS:
  if token in R_NOISING_MAP.keys():
    NOISED_TOKENS.append(token)
  if token in R_COMPOUND_MAP.keys():
    COMPOUNDED_TOKENS.append(token)

prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
if word_order == "sov":
  prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
elif word_order == "svo":
  prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
elif word_order == "vos":
  prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
elif word_order == "vso":
  prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
prompt += "The following is a list of word translations from English to␣
↪Exurbanta:\n"

ALREADY_TRANSLATED = []

# Augmented with random ordering
random.shuffle(COMPOUNDED_TOKENS)
random.shuffle(NOISED_TOKENS)

for token in COMPOUNDED_TOKENS:
  if token not in ALREADY_TRANSLATED:
    ALREADY_TRANSLATED.append(token)
    prompt += f'"{R_COMPOUND_MAP[token][0]} {R_COMPOUND_MAP[token][1]}" means␣
↪"{token}"\n'

for token in NOISED_TOKENS:
  if token not in ALREADY_TRANSLATED:
    ALREADY_TRANSLATED.append(token)
    prompt += f'"{R_NOISING_MAP[token]}" means "{token}"\n'

prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
return prompt
```

```
[ ]: def EXPERIMENT_AB_bilingual_prompting(input_file, label_files, noising_map,␣
↪compound_map, log_dir, result_dir, log_fname = "", result_fname = ""):
  ### Start of Explanation
```

```python
# This code is used to perform bilingual prompting experiment
# ONLY FOR EXPERIMENT WITH CODE 'AB'
### End of Explanation

# Start of Experiment Preparation
import os
from tqdm import tqdm
os.makedirs(log_dir, exist_ok=True)
os.makedirs(result_dir, exist_ok=True)

## Get Experiment Details
EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'bilingual_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'bilingual_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file in label_files:

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = AB_bilingual_prompting(input_sentence, label_sentences[idx],␣
↪noising_map, compound_map, EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
    # temp_log += f"{result}\n"
    temp_log += f"=====\n"

    temp_out = f"$$$ Entry {idx}\n"
    temp_out += f"{result}\n"
    temp_out += f"--ENDOFENTRY--\n"


    if log_fname == "":
      with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
        f.write(temp_log)
    else:
      with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
        f.write(temp_log)

    if result_fname == "":
      with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
        f.write(temp_out)
    else:
      with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
        f.write(temp_out)
```

```
[ ]: EXPERIMENT_AB_bilingual_prompting(INPUT_FILE,
                                   LABEL_FILES,
                                   NOISING_MAP,
                                   COMPOUND_MAP,
                                   LOG_DIR,
                                   RESULT_DIR,
                                   log_fname = "randomized_order",
                                   result_fname = "")
```

```
name = AB
log_dir = experiment_prompts/AB/bilingual_prompting
res_dir = experiment_results/AB/bilingual_prompting
```

```
100%|      | 1012/1012 [00:03<00:00, 334.46it/s]
100%|      | 1012/1012 [00:03<00:00, 322.03it/s]
100%|      | 1012/1012 [00:03<00:00, 260.90it/s]
100%|      | 1012/1012 [00:02<00:00, 348.11it/s]
```

2.1.2 C/DxBA

```
[ ]: def tokenize(text):
       import re
       import string
       ret = []
       for token in text.split(' '):
         result_list = re.findall(r'\w+|[^\w\s]', token)
         flag = 0
         for token in result_list:
           if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
             flag = 1
         if flag:
           ret.append(''.join(result_list))
         elif len(result_list) > 2:
           ret.append(''.join(result_list))
         else:
           ret.extend(result_list)
       return ret

     import random
     random.seed(2023)
     def CDBA_bilingual_prompting(input_sentence, label_sentence, translate_map,␣
      ↪compound_map, noising_map, word_order):
       ### Start of Explanation
       # Helper function for EXPERIMENT_CDBA_bilingual_prompting
       # Uses the tokenize function above (tbh, they are all the same and unchanged)
       ### End of Explanation

       # 0. Prepare prompt
       prompt = "Exurbanta is a lost language to humanity that was found only a few␣
      ↪days ago.\n"
       if word_order == "sov":
         prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
       elif word_order == "svo":
         prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
       elif word_order == "vos":
         prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
       elif word_order == "vso":
```

```
  prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
prompt += "The following is a list of word translations from English to␣
↪Exurbanta:\n"

ALREADY_TRANSLATED = []

INPUT_TOKENS = tokenize(input_sentence)
random.shuffle(INPUT_TOKENS)
# 1. Original --> C/Dx
for token in INPUT_TOKENS:
  if token in translate_map:
    if token not in ALREADY_TRANSLATED:
      ALREADY_TRANSLATED.append(token)
      ALREADY_TRANSLATED.append(TRANSLATE_MAP[token])

      if TRANSLATE_MAP[token] in NOISING_MAP:
        prompt += f'"{token}" means "{NOISING_MAP[TRANSLATE_MAP[token]]}"\n'
        ALREADY_TRANSLATED.append(NOISING_MAP[TRANSLATE_MAP[token]])
      else:
        prompt += f'"{token}" means "{TRANSLATE_MAP[token]}"\n'


# 2. C/Dx --> C/DxB --> C/DxBA
## Note, Even though the A + B experiment is named AB,
## It actually perform compounding first, THEN noising
## Just like this one.
LABEL_TOKENS = tokenize(label_sentence)

R_COMPOUND_MAP = {}
for k,v in compound_map.items():
  R_COMPOUND_MAP[v] = k

R_NOISING_MAP = {}
for k,v in noising_map.items():
  R_NOISING_MAP[v] = k

COMPOUNDED_TOKENS = []
NOISED_TOKENS = []

for token in LABEL_TOKENS:
  if token in R_NOISING_MAP.keys():
    NOISED_TOKENS.append(token)
    if token in R_COMPOUND_MAP.keys():
      COMPOUNDED_TOKENS.append(token)

random.shuffle(COMPOUNDED_TOKENS)
random.shuffle(NOISED_TOKENS)
```

```
  for token in COMPOUNDED_TOKENS:
    if token not in ALREADY_TRANSLATED:
      ALREADY_TRANSLATED.append(token)
      prompt += f'"{R_COMPOUND_MAP[token][0]} {R_COMPOUND_MAP[token][1]}" means␣
↪"{token}"\n'

  for token in NOISED_TOKENS:
    if token not in ALREADY_TRANSLATED:
      ALREADY_TRANSLATED.append(token)
      prompt += f'"{R_NOISING_MAP[token]}" means "{token}"\n'

  prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
  return prompt
```

[ ]:
```
def EXPERIMENT_CDBA_bilingual_prompting(input_file, label_files, translate_map,␣
↪compound_map, noising_map, log_dir, result_dir, log_fname = "", result_fname␣
↪= ""):
  ### Start of Explanation
  # This code is used to perform bilingual prompting experiment
  # ONLY FOR EXPERIMENT WITH CODE 'CDBA'
  ### End of Explanation

  # Start of Experiment Preparation
  import os
  from tqdm import tqdm
  os.makedirs(log_dir, exist_ok=True)
  os.makedirs(result_dir, exist_ok=True)

  ## Get Experiment Details
  EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
  EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
  EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

  os.makedirs(EXPERIMENT_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_resdir, exist_ok=True)

  EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'bilingual_prompting')
  EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'bilingual_prompting')

  os.makedirs(EXPERIMENT_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_resdir, exist_ok=True)

  ### Start of Debug
  print(f"name = {EXPERIMENT_name}")
  print(f"log_dir = {EXPERIMENT_logdir}")
  print(f"res_dir = {EXPERIMENT_resdir}")
```

```
### End of Debug

input_sentences = load_file(input_file)
for label_file in label_files:

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = CDBA_bilingual_prompting(input_sentence, label_sentences[idx],␣
↪translate_map, compound_map, noising_map, EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
    # temp_log += f"{result}\n"
    temp_log += f"=====\n"

    temp_out = f"$$$ Entry {idx}\n"
    temp_out += f"{result}\n"
    temp_out += f"--ENDOFENTRY--\n"

    if log_fname == "":
      with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
        f.write(temp_log)
    else:
      with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
        f.write(temp_log)
      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

D1

[ ]:
```
import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_sov'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_svo'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vos'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vso')
]

TRANSLATE_MAP = load_map(os.path.join(WORK_DIR, 'D1_mapping.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'D1B/D1B_compound_map.pickle'))
NOISING_MAP = load_map(os.path.join(WORK_DIR, 'D1BA/D1BA_noising_map.pickle'))

EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

[ ]:
```
## D1BA - bilingual prompting
EXPERIMENT_CDBA_bilingual_prompting(input_file = INPUT_FILE,
                                    label_files = LABEL_FILES,
                                    translate_map = TRANSLATE_MAP,
                                    compound_map = COMPOUND_MAP,
                                    noising_map = NOISING_MAP,
                                    log_dir = LOG_DIR,
                                    result_dir = RESULT_DIR,
                                    log_fname = "randomized_order",
                                    result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/bilingual_prompting
res_dir = experiment_results/D1BA/bilingual_prompting
```

```
100%|      | 1012/1012 [00:04<00:00, 232.14it/s]
100%|      | 1012/1012 [00:05<00:00, 176.47it/s]
100%|      | 1012/1012 [00:04<00:00, 238.54it/s]
100%|      | 1012/1012 [00:04<00:00, 221.12it/s]
```

2.2 2. Bilingual Mapping with common/rare words removed

2.3 3. Fewshot Prompting, but dictionary entries first, then fewshot examples

2.3.1 AB

[ ]:
```
def tokenize(text):
  import re
  import string
  ret = []
  for token in text.split(' '):
    result_list = re.findall(r'\w+|[^\w\s]', token)
    flag = 0
    for token in result_list:
      if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

def get_index(in_token, few_shots):
  for idx, tokens in few_shots:
    if in_token in tokens:
      return idx
  return in_token

import random
random.seed(2023)
```

```python
def AB_fewshot_prompting(input_sentence, fewshot_input_file,␣
↪fewshot_label_file, noising_map, word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_AB_fewshot_prompting
  # Uses the tokenize function above
  # This functiosn only require the input_sentence
  # AB.1 and AB.2 had to use label_sentence because of the existance of␣
↪compounding words
  ### End of Explanation

  fewshot_input_sentences = []
  with open(fewshot_input_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_input_sentences.append((i, line.strip()))
      i += 1

  fewshot_label_sentences = []
  with open(fewshot_label_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_label_sentences.append((i, line.strip()))
      i += 1

  input_tokens = tokenize(input_sentence)
  few_shot_indexes = []
  unfound_word = []
  for input_token in input_tokens:
    idx = get_index(input_token, fewshot_input_sentences)
    if isinstance(idx, int):
      few_shot_indexes.append(get_index(input_token, fewshot_input_sentences))
    elif isinstance(idx, str):
      unfound_word.append(idx)

  few_shot_indexes = list(set(few_shot_indexes))
  unfound_word = list(set(unfound_word))

  prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
  if word_order == "sov":
    prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
  elif word_order == "svo":
    prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
  elif word_order == "vos":
    prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
  elif word_order == "vso":
    prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
  prompt += f"The following is a list of sentence translations from English to␣
↪Exurbanta:\n"

  for w in unfound_word:
    prompt += f"\n"
    prompt += f"English: {w}\n"
    prompt += f"Exurbanta: {noising_map.get(w, w)}\n"

  for idx in few_shot_indexes:
    prompt += f"\n"
    prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
    prompt += f"Exurbanta: {fewshot_label_sentences[idx-1][1]}\n"

  prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
  return prompt
```

```
[ ]: def EXPERIMENT_AB_fewshot_prompting(input_file, label_files,␣
↪fewshot_input_file, fewshot_label_files, noising_map, log_dir, result_dir,␣
↪log_fname = "", result_fname = ""):
  ### Start of Explanation
  # This code is used to perform fewshot prompting experiment
  # ONLY FOR EXPERIMENT WITH CODE 'AB'
  ### End of Explanation

  # Start of Experiment Preparation
  import os
  from tqdm import tqdm
  os.makedirs(log_dir, exist_ok=True)
  os.makedirs(result_dir, exist_ok=True)

  ## Get Experiment Details
  EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
  EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
  EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

  os.makedirs(EXPERIMENT_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_resdir, exist_ok=True)

  EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
  EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

  os.makedirs(EXPERIMENT_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_resdir, exist_ok=True)

  ### Start of Debug
  print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = AB_fewshot_prompting(input_sentence, fewshot_input_file,␣
↪fewshot_label_file, noising_map, EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
    # temp_log += f"{result}\n"
    temp_log += f"=====\n"

    temp_out = f"$$$ Entry {idx}\n"
    temp_out += f"{result}\n"
    temp_out += f"--ENDOFENTRY--\n"


    if log_fname == "":
      with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
        f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

[ ]:
```
import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vso')
]

NOISING_MAP = load_map(os.path.join(WORK_DIR, 'AB/AB_noising_map.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'B/B_compound_map.pickle'))

EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

[ ]:
```
EXPERIMENT_AB_fewshot_prompting(
    INPUT_FILE,
    LABEL_FILES,
    FEWSHOT_INPUT_FILE,
    FEWSHOT_LABEL_FILES,
    NOISING_MAP,
    LOG_DIR,
    RESULT_DIR,
    log_fname = "entry_then_fewshow",
    result_fname = ""
)
```

```
name = AB
log_dir = experiment_prompts/AB/fewshot_prompting
res_dir = experiment_results/AB/fewshot_prompting
```

```
100%|      | 1012/1012 [00:04<00:00, 232.09it/s]
100%|      | 1012/1012 [00:03<00:00, 281.15it/s]
100%|      | 1012/1012 [00:03<00:00, 284.60it/s]
100%|      | 1012/1012 [00:04<00:00, 214.97it/s]
```

2.3.2 C/DxBA

```
[ ]: def tokenize(text):
       import re
       import string
       ret = []
       for token in text.split(' '):
         result_list = re.findall(r'\w+|[^\w\s]', token)
         flag = 0
         for token in result_list:
           if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
             flag = 1
         if flag:
           ret.append(''.join(result_list))
         elif len(result_list) > 2:
           ret.append(''.join(result_list))
         else:
           ret.extend(result_list)
       return ret

     def get_index(in_token, few_shots):
       for idx, tokens in few_shots:
         if in_token in tokens:
           return idx
       return in_token

     import random
     random.seed(2023)
     def CDBA_fewshot_prompting(input_sentence, label_sentence, fewshot_input_file,␣
       ↪fewshot_label_file, translation_map, compound_map, noising_map, word_order):
```

```
### Start of Explanation
# Helper function for EXPERIMENT_CDBA_fewshot_prompting
# Uses the tokenize function above
# This functiosn only require the input_sentence
# Other prompting functions had to use label_sentence because of the␣
↪existance of compounding words
### End of Explanation

fewshot_input_sentences = []
with open(fewshot_input_file, 'r') as f:
  i = 1
  for line in f:
    fewshot_input_sentences.append((i, line.strip()))
    i += 1

fewshot_label_sentences = []
with open(fewshot_label_file, 'r') as f:
  i = 1
  for line in f:
    fewshot_label_sentences.append((i, line.strip()))
    i += 1

input_tokens = tokenize(input_sentence)
few_shot_indexes = []
unfound_en_word = []
unfound_exurbanta_word = []
for input_token in input_tokens:
  idx = get_index(input_token, fewshot_input_sentences)
  if isinstance(idx, int):
    if idx not in few_shot_indexes:
      few_shot_indexes.append(idx)
  elif isinstance(idx, str):
    if idx not in unfound_en_word:
      unfound_en_word.append(idx)

label_tokens = tokenize(label_sentence)
for label_token in label_tokens:
  idx = get_index(label_token, fewshot_label_sentences)
  if isinstance(idx, int):
    if idx not in few_shot_indexes:
      few_shot_indexes.append(idx)
  elif isinstance(idx, str):
    if idx not in unfound_exurbanta_word:
      unfound_exurbanta_word.append(idx)

prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
```

```
if word_order == "sov":
  prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
elif word_order == "svo":
  prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
elif word_order == "vos":
  prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
elif word_order == "vso":
  prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
prompt += f"The following is a list of sentence translations from English to␣
↪Exurbanta:\n"

ALREADY_TRANSLATED = []
# Handles unhandled Noising
for w in unfound_en_word:
  if w not in ALREADY_TRANSLATED:
    prompt += f"\n"
    prompt += f"English: {w}\n"
    if w in noising_map:
      ALREADY_TRANSLATED.append(noising_map[w])
      prompt += f"Exurbanta: {noising_map[w]}\n"
    else:
      prompt += f"Exurbanta: {w}\n"
    ALREADY_TRANSLATED.append(w)

# Handles unhandled Compounding
R_COMPOUND_MAP = {}
for k,v in compound_map.items():
  R_COMPOUND_MAP[v] = k

for w in unfound_exurbanta_word:
  if w not in ALREADY_TRANSLATED:
    prompt += f"\n"
    if w in R_COMPOUND_MAP:
      prompt += f"English: {R_COMPOUND_MAP[w][0]} {R_COMPOUND_MAP[w][1]}\n"
      prompt += f"Exurbanta: {w}\n"
      ALREADY_TRANSLATED.append(w)

for idx in few_shot_indexes:
  prompt += f"\n"
  prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
  prompt += f"Exurbanta: {fewshot_label_sentences[idx-1][1]}\n"

prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
return prompt
```

```
[ ]: def EXPERIMENT_CDBA_fewshot_prompting(input_file, label_files,␣
      ↪fewshot_input_file, fewshot_label_files, translate_map, compound_map,␣
      ↪noising_map, log_dir, result_dir, log_fname = "", result_fname = ""):
       ### Start of Explanation
       # This code is used to perform fewshot prompting experiment
       # ONLY FOR EXPERIMENT WITH CODE 'AB'
       ### End of Explanation

       # Start of Experiment Preparation
       import os
       from tqdm import tqdm
       os.makedirs(log_dir, exist_ok=True)
       os.makedirs(result_dir, exist_ok=True)

       ## Get Experiment Details
       EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
       EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
       EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

       os.makedirs(EXPERIMENT_logdir, exist_ok=True)
       os.makedirs(EXPERIMENT_resdir, exist_ok=True)

       EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
       EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

       os.makedirs(EXPERIMENT_logdir, exist_ok=True)
       os.makedirs(EXPERIMENT_resdir, exist_ok=True)

       ### Start of Debug
       print(f"name = {EXPERIMENT_name}")
       print(f"log_dir = {EXPERIMENT_logdir}")
       print(f"res_dir = {EXPERIMENT_resdir}")
       ### End of Debug

       input_sentences = load_file(input_file)
       for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

         ## Create folder preparations
         EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
         EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
         EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
         os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
         os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
         ## End of Experiment Preparation

         # Start Experiment
         label_sentences = load_file(label_file)
```

```
    assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

    prompts = []
    for idx, input_sentence in enumerate(tqdm(input_sentences)):
      prompt = CDBA_fewshot_prompting(input_sentence,
                                      label_sentences[idx],
                                      fewshot_input_file,
                                      fewshot_label_file,
                                      translate_map,
                                      compound_map,
                                      noising_map,
                                      EXPERIMENT_order)
      dialog = [
          {'role': 'system', 'content': 'You can only use one sentence.'},
          {'role': 'user', 'content': prompt}
      ]
      payload = {
        "inputs": [dialog],
        "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
      }
      # result = query_endpoint(payload)[0]['generation']['content']
      result = 'test'
      # temp_log = f"$$$ Entry {idx}\n"
      temp_log = f"{prompt}\n"
      # temp_log += f"{result}\n"
      temp_log += f"=====\n"

      temp_out = f"$$$ Entry {idx}\n"
      temp_out += f"{result}\n"
      temp_out += f"--ENDOFENTRY--\n"


      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

D1

```
[ ]: import os

     WORK_DIR = "/content/extract"

     INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
     FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

     LABEL_FILES = [
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_sov'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_svo'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vos'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vso')
     ]

     FEWSHOT_LABEL_FILES = [
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_sov'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_svo'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vos'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vso')
     ]

     TRANSLATE_MAP = load_map(os.path.join(WORK_DIR, 'D1_mapping.pickle'))
     COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'D1B/D1B_compound_map.pickle'))
     NOISING_MAP = load_map(os.path.join(WORK_DIR, 'D1BA/D1BA_noising_map.pickle'))

     LOG_DIR = "experiment_prompts/"
     RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_CDBA_fewshot_prompting(input_file = INPUT_FILE,
                                       label_files = LABEL_FILES,
                                       fewshot_input_file = FEWSHOT_INPUT_FILE,
                                       fewshot_label_files = FEWSHOT_LABEL_FILES,
                                       translate_map = TRANSLATE_MAP,
                                       compound_map = COMPOUND_MAP,
                                       noising_map = NOISING_MAP,
                                       log_dir = LOG_DIR,
                                       result_dir = RESULT_DIR,
                                       log_fname = "entry_then_fewshow",
                                       result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/fewshot_prompting
res_dir = experiment_results/D1BA/fewshot_prompting

100%|      | 1012/1012 [00:07<00:00, 144.36it/s]
100%|      | 1012/1012 [00:08<00:00, 125.51it/s]
100%|      | 1012/1012 [00:07<00:00, 141.64it/s]
```

```
100%|      | 1012/1012 [00:08<00:00, 125.44it/s]
```

2.4 4. Fewshot Prompting, but only give 5 random examples

2.4.1 AB

```
[ ]: def tokenize(text):
       import re
       import string
       ret = []
       for token in text.split(' '):
         result_list = re.findall(r'\w+|[^\w\s]', token)
         flag = 0
         for token in result_list:
           if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
             flag = 1
         if flag:
           ret.append(''.join(result_list))
         elif len(result_list) > 2:
           ret.append(''.join(result_list))
         else:
           ret.extend(result_list)
       return ret

     def get_index(in_token, few_shots):
       for idx, tokens in few_shots:
         if in_token in tokens:
           return idx
       return in_token

     import random
     random.seed(2023)
     def AB_fiveshot_prompting(input_sentence, fewshot_input_file,␣
      ↪fewshot_label_file, noising_map, word_order):
       ### Start of Explanation
       # Helper function for EXPERIMENT_AB_fewshot_prompting
       # Uses the tokenize function above
       # This functiosn only require the input_sentence
       # AB.1 and AB.2 had to use label_sentence because of the existance of␣
      ↪compounding words
       ### End of Explanation

       fewshot_input_sentences = []
       with open(fewshot_input_file, 'r') as f:
         i = 1
         for line in f:
           fewshot_input_sentences.append((i, line.strip()))
           i += 1
```

```
fewshot_label_sentences = []
with open(fewshot_label_file, 'r') as f:
  i = 1
  for line in f:
    fewshot_label_sentences.append((i, line.strip()))
    i += 1


few_shot_indexes = random.sample(range(1, 998), 5)

prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
if word_order == "sov":
  prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
elif word_order == "svo":
  prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
elif word_order == "vos":
  prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
elif word_order == "vso":
  prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
prompt += f"The following is a list of sentence translations from English to␣
↪Exurbanta:\n"

for idx in few_shot_indexes:
  prompt += f"\n"
  prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
  prompt += f"Exurbanta: {fewshot_label_sentences[idx-1][1]}\n"

prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
return prompt
```

```
[ ]: def EXPERIMENT_AB_fiveshot_prompting(input_file, label_files,␣
↪fewshot_input_file, fewshot_label_files, noising_map, log_dir, result_dir,␣
↪log_fname = "", result_fname = ""):
  ### Start of Explanation
  # This code is used to perform fewshot prompting experiment
  # ONLY FOR EXPERIMENT WITH CODE 'AB'
  ### End of Explanation

  # Start of Experiment Preparation
  import os
  from tqdm import tqdm
  os.makedirs(log_dir, exist_ok=True)
  os.makedirs(result_dir, exist_ok=True)
```

```
## Get Experiment Details
EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = AB_fiveshot_prompting(input_sentence, fewshot_input_file,␣
↪fewshot_label_file, noising_map, EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
      }
      # result = query_endpoint(payload)[0]['generation']['content']
      result = "test"
      # temp_log = f"$$$ Entry {idx}\n"
      temp_log = f"{prompt}\n"
      # temp_log += f"{result}\n"
      temp_log += f"=====\n"

      temp_out = f"$$$ Entry {idx}\n"
      temp_out += f"{result}\n"
      temp_out += f"--ENDOFENTRY--\n"


      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

```
[ ]: import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vso')
]

NOISING_MAP = load_map(os.path.join(WORK_DIR, 'AB/AB_noising_map.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'B/B_compound_map.pickle'))

EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_AB_fiveshot_prompting(
         INPUT_FILE,
         LABEL_FILES,
         FEWSHOT_INPUT_FILE,
         FEWSHOT_LABEL_FILES,
         NOISING_MAP,
         LOG_DIR,
         RESULT_DIR,
         log_fname = "fiveshot",
         result_fname = ""
     )
```

```
name = AB
log_dir = experiment_prompts/AB/fewshot_prompting
res_dir = experiment_results/AB/fewshot_prompting

100%|      | 1012/1012 [00:01<00:00, 626.42it/s]
100%|      | 1012/1012 [00:01<00:00, 602.44it/s]
100%|      | 1012/1012 [00:02<00:00, 432.94it/s]
100%|      | 1012/1012 [00:02<00:00, 489.77it/s]
```

2.4.2 C/DxBA

```
[ ]: def tokenize(text):
       import re
       import string
       ret = []
       for token in text.split(' '):
         result_list = re.findall(r'\w+|[^\w\s]', token)
         flag = 0
         for token in result_list:
           if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
             flag = 1
         if flag:
           ret.append(''.join(result_list))
```

```
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

def get_index(in_token, few_shots):
  for idx, tokens in few_shots:
    if in_token in tokens:
      return idx
  return in_token

import random
random.seed(2023)
def CDBA_fiveshot_prompting(input_sentence, label_sentence, fewshot_input_file,␣
 ↪fewshot_label_file, translation_map, compound_map, noising_map, word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_CDBA_fewshot_prompting
  # Uses the tokenize function above
  # This functiosn only require the input_sentence
  # Other prompting functions had to use label_sentence because of the␣
 ↪existance of compounding words
  ### End of Explanation
  fewshot_input_sentences = []
  with open(fewshot_input_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_input_sentences.append((i, line.strip()))
      i += 1

  fewshot_label_sentences = []
  with open(fewshot_label_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_label_sentences.append((i, line.strip()))
      i += 1

  few_shot_indexes = random.sample(range(1, 998), 5)

  prompt = "Exurbanta is a lost language to humanity that was found only a few␣
 ↪days ago.\n"
  if word_order == "sov":
    prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
  elif word_order == "svo":
    prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
  elif word_order == "vos":
    prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
  elif word_order == "vso":
    prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
  prompt += f"The following is a list of sentence translations from English to␣
↪Exurbanta:\n"

  for idx in few_shot_indexes:
    prompt += f"\n"
    prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
    prompt += f"Exurbanta: {fewshot_label_sentences[idx-1][1]}\n"

  prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
  return prompt
```

```
[ ]: def EXPERIMENT_CDBA_fiveshot_prompting(input_file, label_files,␣
↪fewshot_input_file, fewshot_label_files, translate_map, compound_map,␣
↪noising_map, log_dir, result_dir, log_fname = "", result_fname = ""):
  ### Start of Explanation
  # This code is used to perform fewshot prompting experiment
  # ONLY FOR EXPERIMENT WITH CODE 'AB'
  ### End of Explanation

  # Start of Experiment Preparation
  import os
  from tqdm import tqdm
  os.makedirs(log_dir, exist_ok=True)
  os.makedirs(result_dir, exist_ok=True)

  ## Get Experiment Details
  EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
  EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
  EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

  os.makedirs(EXPERIMENT_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_resdir, exist_ok=True)

  EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
  EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

  os.makedirs(EXPERIMENT_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_resdir, exist_ok=True)

  ### Start of Debug
  print(f"name = {EXPERIMENT_name}")
  print(f"log_dir = {EXPERIMENT_logdir}")
  print(f"res_dir = {EXPERIMENT_resdir}")
  ### End of Debug
```

```
input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = CDBA_fiveshot_prompting(input_sentence,
                                     label_sentences[idx],
                                     fewshot_input_file,
                                     fewshot_label_file,
                                     translate_map,
                                     compound_map,
                                     noising_map,
                                     EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = 'test'
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
    # temp_log += f"{result}\n"
    temp_log += f"=====\n"

    temp_out = f"$$$ Entry {idx}\n"
    temp_out += f"{result}\n"
    temp_out += f"--ENDOFENTRY--\n"
```

```
      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

D1

```
[ ]: import os

     WORK_DIR = "/content/extract"

     INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
     FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

     LABEL_FILES = [
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_sov'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_svo'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vos'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vso')
     ]

     FEWSHOT_LABEL_FILES = [
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_sov'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_svo'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vos'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vso')
     ]

     TRANSLATE_MAP = load_map(os.path.join(WORK_DIR, 'D1_mapping.pickle'))
     COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'D1B/D1B_compound_map.pickle'))
     NOISING_MAP = load_map(os.path.join(WORK_DIR, 'D1BA/D1BA_noising_map.pickle'))

     LOG_DIR = "experiment_prompts/"
     RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_CDBA_fiveshot_prompting(input_file = INPUT_FILE,
                                        label_files = LABEL_FILES,
                                        fewshot_input_file = FEWSHOT_INPUT_FILE,
                                        fewshot_label_files = FEWSHOT_LABEL_FILES,
                                        translate_map = TRANSLATE_MAP,
                                        compound_map = COMPOUND_MAP,
                                        noising_map = NOISING_MAP,
                                        log_dir = LOG_DIR,
                                        result_dir = RESULT_DIR,
                                        log_fname = "fiveshot",
                                        result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/fewshot_prompting
res_dir = experiment_results/D1BA/fewshot_prompting
```

```
100%|      | 1012/1012 [00:01<00:00, 649.82it/s]
100%|      | 1012/1012 [00:01<00:00, 647.43it/s]
100%|      | 1012/1012 [00:01<00:00, 644.76it/s]
100%|      | 1012/1012 [00:01<00:00, 649.94it/s]
```

2.5 5. Fewshot Prompting, but, change a couple of target word from fewshot examples to become a dictionary entries instead.

2.5.1 AB

[ ]:
```
def tokenize(text):
  import re
  import string
  ret = []
  for token in text.split(' '):
    result_list = re.findall(r'\w+|[^\w\s]', token)
    flag = 0
    for token in result_list:
      if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

def get_index(in_token, few_shots):
  for idx, tokens in few_shots:
    if in_token in tokens:
      return (in_token, idx)
  return in_token
```

```
import random
random.seed(2023)
def AB_mixed_prompting(input_sentence, fewshot_input_file, fewshot_label_file,␣
 ↪noising_map, word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_AB_fewshot_prompting
  # Uses the tokenize function above
  # This functiosn only require the input_sentence
  # AB.1 and AB.2 had to use label_sentence because of the existance of␣
 ↪compounding words
  ### End of Explanation

  fewshot_input_sentences = []
  with open(fewshot_input_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_input_sentences.append((i, line.strip()))
      i += 1

  fewshot_label_sentences = []
  with open(fewshot_label_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_label_sentences.append((i, line.strip()))
      i += 1

  input_tokens = tokenize(input_sentence)
  few_shot_indexes = []
  unfound_word = []
  for input_token in input_tokens:
    idx = get_index(input_token, fewshot_input_sentences)
    if isinstance(idx, tuple):
      few_shot_indexes.append(get_index(input_token, fewshot_input_sentences))
    elif isinstance(idx, str):
      unfound_word.append(idx)

  few_shot_indexes = list(set(few_shot_indexes))
  unfound_word = list(set(unfound_word))

  random.shuffle(few_shot_indexes)

  half_len = len(few_shot_indexes)//2
  temp_shots = few_shot_indexes[:half_len]
  few_shot_indexes = few_shot_indexes[half_len:]

  for token, idx in temp_shots:
    unfound_word.append(token)
    prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
    if word_order == "sov":
      prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
    elif word_order == "svo":
      prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
    elif word_order == "vos":
      prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
    elif word_order == "vso":
      prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
    prompt += f"The following is a list of sentence translations from English to␣
↪Exurbanta:\n"

    for w in unfound_word:
      prompt += f"\n"
      prompt += f"English: {w}\n"
      prompt += f"Exurbanta: {noising_map.get(w, w)}\n"

    for token, idx in few_shot_indexes:
      prompt += f"\n"
      prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
      prompt += f"Exurbanta: {fewshot_label_sentences[idx-1][1]}\n"

    prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
    return prompt
```

```
[ ]: def EXPERIMENT_AB_mixed_prompting(input_file, label_files, fewshot_input_file,␣
↪fewshot_label_files, noising_map, log_dir, result_dir, log_fname = "",␣
↪result_fname = ""):
    ### Start of Explanation
    # This code is used to perform fewshot prompting experiment
    # ONLY FOR EXPERIMENT WITH CODE 'AB'
    ### End of Explanation

    # Start of Experiment Preparation
    import os
    from tqdm import tqdm
    os.makedirs(log_dir, exist_ok=True)
    os.makedirs(result_dir, exist_ok=True)

    ## Get Experiment Details
    EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
    EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
    EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)
```

```
os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = AB_mixed_prompting(input_sentence, fewshot_input_file,␣
↪fewshot_label_file, noising_map, EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
```

```
      # temp_log += f"{result}\n"
      temp_log += f"=====\n"

      temp_out = f"$$$ Entry {idx}\n"
      temp_out += f"{result}\n"
      temp_out += f"--ENDOFENTRY--\n"


      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

```
[ ]: import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vso')
]

NOISING_MAP = load_map(os.path.join(WORK_DIR, 'AB/AB_noising_map.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'B/B_compound_map.pickle'))
```

```
EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_AB_mixed_prompting(
         INPUT_FILE,
         LABEL_FILES,
         FEWSHOT_INPUT_FILE,
         FEWSHOT_LABEL_FILES,
         NOISING_MAP,
         LOG_DIR,
         RESULT_DIR,
         log_fname = "mixed",
         result_fname = ""
     )
```

```
name = AB
log_dir = experiment_prompts/AB/fewshot_prompting
res_dir = experiment_results/AB/fewshot_prompting

100%|      | 1012/1012 [00:04<00:00, 233.47it/s]
100%|      | 1012/1012 [00:04<00:00, 249.05it/s]
100%|      | 1012/1012 [00:03<00:00, 279.19it/s]
100%|      | 1012/1012 [00:03<00:00, 271.71it/s]
```

2.5.2 C/DxBA

```
[ ]: def tokenize(text):
       import re
       import string
       ret = []
       for token in text.split(' '):
         result_list = re.findall(r'\w+|[^\w\s]', token)
         flag = 0
         for token in result_list:
           if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
             flag = 1
         if flag:
           ret.append(''.join(result_list))
         elif len(result_list) > 2:
           ret.append(''.join(result_list))
         else:
           ret.extend(result_list)
       return ret
```

```
def get_index(in_token, few_shots):
  for idx, tokens in few_shots:
    if in_token in tokens:
      return (in_token, idx)
  return in_token

import random
random.seed(2023)
def CDBA_mixed_prompting(input_sentence, label_sentence, fewshot_input_file,␣
↪fewshot_label_file, translation_map, compound_map, noising_map, word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_CDBA_fewshot_prompting
  # Uses the tokenize function above
  # This functiosn only require the input_sentence
  # Other prompting functions had to use label_sentence because of the␣
↪existance of compounding words
  ### End of Explanation

  fewshot_input_sentences = []
  with open(fewshot_input_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_input_sentences.append((i, line.strip()))
      i += 1

  fewshot_label_sentences = []
  with open(fewshot_label_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_label_sentences.append((i, line.strip()))
      i += 1

  input_tokens = tokenize(input_sentence)
  few_shot_indexes_en = []
  few_shot_indexes_exurbanta = []
  unfound_en_word = []
  unfound_exurbanta_word = []
  for input_token in input_tokens:
    idx = get_index(input_token, fewshot_input_sentences)
    if isinstance(idx, tuple):
      if idx not in few_shot_indexes_en:
        few_shot_indexes_en.append(idx)
    elif isinstance(idx, str):
      if idx not in unfound_en_word:
        unfound_en_word.append(idx)
```

```
label_tokens = tokenize(label_sentence)
for label_token in label_tokens:
  idx = get_index(label_token, fewshot_label_sentences)
  if isinstance(idx, tuple):
    if idx not in few_shot_indexes_exurbanta:
      few_shot_indexes_exurbanta.append(idx)
  elif isinstance(idx, str):
    if idx not in unfound_exurbanta_word:
      unfound_exurbanta_word.append(idx)

few_shot_indexes = []
random.shuffle(few_shot_indexes_en)

half_len = len(few_shot_indexes_en)//2
temp_shots = few_shot_indexes_en[:half_len]
few_shot_indexes = few_shot_indexes[half_len:]

for token, idx in temp_shots:
  unfound_en_word.append(token)

random.shuffle(few_shot_indexes_exurbanta)

half_len = len(few_shot_indexes_exurbanta)//2
temp_shots = few_shot_indexes_exurbanta[:half_len]
few_shot_indexes += few_shot_indexes_exurbanta[half_len:]

for token, idx in temp_shots:
  unfound_exurbanta_word.append(token)

prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
if word_order == "sov":
  prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
elif word_order == "svo":
  prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
elif word_order == "vos":
  prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
elif word_order == "vso":
  prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
prompt += f"The following is a list of sentence translations from English to␣
↪Exurbanta:\n"

ALREADY_TRANSLATED = []
# Handles unhandled Noising
for w in unfound_en_word:
  if w not in ALREADY_TRANSLATED:
    prompt += f"\n"
```

```
      prompt += f"English: {w}\n"
      if w in noising_map:
        ALREADY_TRANSLATED.append(noising_map[w])
        prompt += f"Exurbanta: {noising_map[w]}\n"
      else:
        prompt += f"Exurbanta: {w}\n"
      ALREADY_TRANSLATED.append(w)

  # Handles unhandled Compounding
  R_COMPOUND_MAP = {}
  for k,v in compound_map.items():
    R_COMPOUND_MAP[v] = k

  for w in unfound_exurbanta_word:
    if w not in ALREADY_TRANSLATED:
      if w in R_COMPOUND_MAP:
        prompt += f"\n"
        prompt += f"English: {R_COMPOUND_MAP[w][0]} {R_COMPOUND_MAP[w][1]}\n"
        prompt += f"Exurbanta: {w}\n"
        ALREADY_TRANSLATED.append(w)

  for token, idx in few_shot_indexes:
    prompt += f"\n"
    prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
    prompt += f"Exurbanta: {fewshot_label_sentences[idx-1][1]}\n"

  prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
  return prompt
```

```
[ ]: def EXPERIMENT_CDBA_mixed_prompting(input_file, label_files,␣
  ↪fewshot_input_file, fewshot_label_files, translate_map, compound_map,␣
  ↪noising_map, log_dir, result_dir, log_fname = "", result_fname = ""):
  ### Start of Explanation
  # This code is used to perform fewshot prompting experiment
  # ONLY FOR EXPERIMENT WITH CODE 'AB'
  ### End of Explanation

  # Start of Experiment Preparation
  import os
  from tqdm import tqdm
  os.makedirs(log_dir, exist_ok=True)
  os.makedirs(result_dir, exist_ok=True)

  ## Get Experiment Details
  EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
  EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
```

```
EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = CDBA_mixed_prompting(input_sentence,
                                  label_sentences[idx],
                                  fewshot_input_file,
                                  fewshot_label_file,
                                  translate_map,
                                  compound_map,
                                  noising_map,
                                  EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
```

```
      payload = {
        "inputs": [dialog],
        "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
      }
      # result = query_endpoint(payload)[0]['generation']['content']
      result = 'test'
      # temp_log = f"$$$ Entry {idx}\n"
      temp_log = f"{prompt}\n"
      # temp_log += f"{result}\n"
      temp_log += f"=====\n"

      temp_out = f"$$$ Entry {idx}\n"
      temp_out += f"{result}\n"
      temp_out += f"--ENDOFENTRY--\n"


      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

D1

[ ]:
```
import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_sov'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_svo'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vos'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vso')
]
```

```
FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vso')
]

TRANSLATE_MAP = load_map(os.path.join(WORK_DIR, 'D1_mapping.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'D1B/D1B_compound_map.pickle'))
NOISING_MAP = load_map(os.path.join(WORK_DIR, 'D1BA/D1BA_noising_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_CDBA_mixed_prompting(input_file = INPUT_FILE,
                                  label_files = LABEL_FILES,
                                  fewshot_input_file = FEWSHOT_INPUT_FILE,
                                  fewshot_label_files = FEWSHOT_LABEL_FILES,
                                  translate_map = TRANSLATE_MAP,
                                  compound_map = COMPOUND_MAP,
                                  noising_map = NOISING_MAP,
                                  log_dir = LOG_DIR,
                                  result_dir = RESULT_DIR,
                                  log_fname = "mixed",
                                  result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/fewshot_prompting
res_dir = experiment_results/D1BA/fewshot_prompting
```

```
100%|          | 1012/1012 [00:08<00:00, 120.92it/s]
100%|          | 1012/1012 [00:08<00:00, 120.22it/s]
100%|          | 1012/1012 [00:07<00:00, 138.90it/s]
100%|          | 1012/1012 [00:08<00:00, 116.73it/s]
```

2.6 6. Kitchen Sink: Gives full bilingual dictionary, then fewshot examples.

2.6.1 AB

```
[ ]: def tokenize(text):
       import re
       import string
       ret = []
       for token in text.split(' '):
         result_list = re.findall(r'\w+|[^\w\s]', token)
         flag = 0
         for token in result_list:
           if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
```

```
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

def get_index(in_token, few_shots):
  for idx, tokens in few_shots:
    if in_token in tokens:
      return idx
  return in_token

import random
random.seed(2023)
def AB_kitchensink_prompting(input_sentence, label_sentence,␣
 ↪fewshot_input_file, fewshot_label_file, compound_map, noising_map,␣
 ↪word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_AB_bilingual_prompting
  # Uses the tokenize function above
  ### End of Explanation
  LABEL_TOKENS = tokenize(label_sentence)
  R_NOISING_MAP = {}
  for k,v in noising_map.items():
    R_NOISING_MAP[v] = k

  R_COMPOUND_MAP = {}
  for k,v in compound_map.items():
    R_COMPOUND_MAP[v] = k

  NOISED_TOKENS = []
  COMPOUNDED_TOKENS = []

  for token in LABEL_TOKENS:
    if token in R_NOISING_MAP.keys():
      NOISED_TOKENS.append(token)
    if token in R_COMPOUND_MAP.keys():
      COMPOUNDED_TOKENS.append(token)

  prompt = "Exurbanta is a lost language to humanity that was found only a few␣
 ↪days ago.\n"
  if word_order == "sov":
    prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
  elif word_order == "svo":
    prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
  elif word_order == "vos":
    prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
  elif word_order == "vso":
    prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
  prompt += "The following is a list of word translations from English to␣
↪Exurbanta:\n"

  ALREADY_TRANSLATED = []

  for token in COMPOUNDED_TOKENS:
    if token not in ALREADY_TRANSLATED:
      ALREADY_TRANSLATED.append(token)
      prompt += f'"{R_COMPOUND_MAP[token][0]} {R_COMPOUND_MAP[token][1]}" means␣
↪"{token}"\n'

  for token in NOISED_TOKENS:
    if token not in ALREADY_TRANSLATED:
      ALREADY_TRANSLATED.append(token)
      prompt += f'"{R_NOISING_MAP[token]}" means "{token}"\n'

  # ========= FEW SHOT ===========
  fewshot_input_sentences = []
  with open(fewshot_input_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_input_sentences.append((i, line.strip()))
      i += 1

  fewshot_label_sentences = []
  with open(fewshot_label_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_label_sentences.append((i, line.strip()))
      i += 1

  input_tokens = tokenize(input_sentence)
  few_shot_indexes = []
  unfound_word = []
  for input_token in input_tokens:
    idx = get_index(input_token, fewshot_input_sentences)
    if isinstance(idx, int):
      few_shot_indexes.append(get_index(input_token, fewshot_input_sentences))
    elif isinstance(idx, str):
      unfound_word.append(idx)

  few_shot_indexes = list(set(few_shot_indexes))
  unfound_word = list(set(unfound_word))

  for idx in few_shot_indexes:
    prompt += f"\n"
    prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
    prompt += f"Exurbanta: {fewshot_label_sentences[idx-1][1]}\n"

  prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
  return prompt
```

```
[ ]: def EXPERIMENT_AB_kitchensink_prompting(input_file, label_files,␣
      ↪fewshot_input_file, fewshot_label_files, noising_map, compound_map, log_dir,␣
      ↪result_dir, log_fname = "", result_fname = ""):
       ### Start of Explanation
       # This code is used to perform fewshot prompting experiment
       # ONLY FOR EXPERIMENT WITH CODE 'AB'
       ### End of Explanation

       # Start of Experiment Preparation
       import os
       from tqdm import tqdm
       os.makedirs(log_dir, exist_ok=True)
       os.makedirs(result_dir, exist_ok=True)

       ## Get Experiment Details
       EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
       EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
       EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

       os.makedirs(EXPERIMENT_logdir, exist_ok=True)
       os.makedirs(EXPERIMENT_resdir, exist_ok=True)

       EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
       EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

       os.makedirs(EXPERIMENT_logdir, exist_ok=True)
       os.makedirs(EXPERIMENT_resdir, exist_ok=True)

       ### Start of Debug
       print(f"name = {EXPERIMENT_name}")
       print(f"log_dir = {EXPERIMENT_logdir}")
       print(f"res_dir = {EXPERIMENT_resdir}")
       ### End of Debug

       input_sentences = load_file(input_file)
       for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):
```

```
  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = AB_kitchensink_prompting(input_sentence = input_sentence,
                                      label_sentence = label_sentences[idx],
                                      fewshot_input_file = fewshot_input_file,
                                      fewshot_label_file = fewshot_label_file,
                                      compound_map = compound_map,
                                      noising_map = noising_map,
                                      word_order = EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
    # temp_log += f"{result}\n"
    temp_log += f"=====\n"

    temp_out = f"$$$ Entry {idx}\n"
    temp_out += f"{result}\n"
    temp_out += f"--ENDOFENTRY--\n"


    if log_fname == "":
      with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
        f.write(temp_log)
    else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

[ ]:
```
import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vso')
]

NOISING_MAP = load_map(os.path.join(WORK_DIR, 'AB/AB_noising_map.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'B/B_compound_map.pickle'))

EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

[ ]:
```
EXPERIMENT_AB_kitchensink_prompting(
    INPUT_FILE,
    LABEL_FILES,
    FEWSHOT_INPUT_FILE,
    FEWSHOT_LABEL_FILES,
    NOISING_MAP,
    COMPOUND_MAP,
    LOG_DIR,
    RESULT_DIR,
    log_fname = "kitchensink",
    result_fname = ""
)
```

```
name = AB
log_dir = experiment_prompts/AB/fewshot_prompting
res_dir = experiment_results/AB/fewshot_prompting
```

```
100%|          | 1012/1012 [00:08<00:00, 117.84it/s]
100%|          | 1012/1012 [00:07<00:00, 140.81it/s]
100%|          | 1012/1012 [00:08<00:00, 121.52it/s]
100%|          | 1012/1012 [00:08<00:00, 123.20it/s]
```

2.6.2 C/DxBA

[ ]:

```
def tokenize(text):
  import re
  import string
  ret = []
  for token in text.split(' '):
    result_list = re.findall(r'\w+|[^\w\s]', token)
    flag = 0
    for token in result_list:
      if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

import random
random.seed(2023)
def CDBA_kitchensink_prompting(input_sentence, label_sentence,␣
 ↪fewshot_input_file, fewshot_label_file, translate_map, compound_map,␣
 ↪noising_map, word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_CDBA_bilingual_prompting
  # Uses the tokenize function above (tbh, they are all the same and unchanged)
  ### End of Explanation
```

```
# 0. Prepare prompt
prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
if word_order == "sov":
  prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
elif word_order == "svo":
  prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
elif word_order == "vos":
  prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
elif word_order == "vso":
  prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
prompt += "The following is a list of word translations from English to␣
↪Exurbanta:\n"

ALREADY_TRANSLATED = []

INPUT_TOKENS = tokenize(input_sentence)
# 1. Original --> C/Dx
for token in INPUT_TOKENS:
  if token in translate_map:
    if token not in ALREADY_TRANSLATED:
      ALREADY_TRANSLATED.append(token)
      ALREADY_TRANSLATED.append(TRANSLATE_MAP[token])

      if TRANSLATE_MAP[token] in NOISING_MAP:
        prompt += f'"{token}" means "{NOISING_MAP[TRANSLATE_MAP[token]]}"\n'
        ALREADY_TRANSLATED.append(NOISING_MAP[TRANSLATE_MAP[token]])
      else:
        prompt += f'"{token}" means "{TRANSLATE_MAP[token]}"\n'


# 2. C/Dx --> C/DxB --> C/DxBA
## Note, Even though the A + B experiment is named AB,
## It actually perform compounding first, THEN noising
## Just like this one.
LABEL_TOKENS = tokenize(label_sentence)

R_COMPOUND_MAP = {}
for k,v in compound_map.items():
  R_COMPOUND_MAP[v] = k

R_NOISING_MAP = {}
for k,v in noising_map.items():
  R_NOISING_MAP[v] = k

COMPOUNDED_TOKENS = []
NOISED_TOKENS = []
```

```
for token in LABEL_TOKENS:
  if token in R_NOISING_MAP.keys():
    NOISED_TOKENS.append(token)
    if token in R_COMPOUND_MAP.keys():
      COMPOUNDED_TOKENS.append(token)

for token in COMPOUNDED_TOKENS:
  if token not in ALREADY_TRANSLATED:
    ALREADY_TRANSLATED.append(token)
    prompt += f'"{R_COMPOUND_MAP[token][0]} {R_COMPOUND_MAP[token][1]}" means␣
↪"{token}"\n'

for token in NOISED_TOKENS:
  if token not in ALREADY_TRANSLATED:
    ALREADY_TRANSLATED.append(token)
    prompt += f'"{R_NOISING_MAP[token]}" means "{token}"\n'

# ====== Fewshot ======
fewshot_input_sentences = []
with open(fewshot_input_file, 'r') as f:
  i = 1
  for line in f:
    fewshot_input_sentences.append((i, line.strip()))
    i += 1

fewshot_label_sentences = []
with open(fewshot_label_file, 'r') as f:
  i = 1
  for line in f:
    fewshot_label_sentences.append((i, line.strip()))
    i += 1

input_tokens = tokenize(input_sentence)
few_shot_indexes = []
unfound_en_word = []
unfound_exurbanta_word = []
for input_token in input_tokens:
  idx = get_index(input_token, fewshot_input_sentences)
  if isinstance(idx, int):
    if idx not in few_shot_indexes:
      few_shot_indexes.append(idx)
  elif isinstance(idx, str):
    if idx not in unfound_en_word:
      unfound_en_word.append(idx)

label_tokens = tokenize(label_sentence)
```

```
  for label_token in label_tokens:
    idx = get_index(label_token, fewshot_label_sentences)
    if isinstance(idx, int):
      if idx not in few_shot_indexes:
        few_shot_indexes.append(idx)
    elif isinstance(idx, str):
      if idx not in unfound_exurbanta_word:
        unfound_exurbanta_word.append(idx)

  ALREADY_TRANSLATED = []
  # Handles unhandled Noising
  # for w in unfound_en_word:
  #   if w not in ALREADY_TRANSLATED:
  #     prompt += f"\n"
  #     prompt += f"English: {w}\n"
  #     if w in noising_map:
  #       ALREADY_TRANSLATED.append(noising_map[w])
  #       prompt += f"Exurbanta: {noising_map[w]}\n"
  #     else:
  #       prompt += f"Exurbanta: {w}\n"
  #     ALREADY_TRANSLATED.append(w)

  # Handles unhandled Compounding
  R_COMPOUND_MAP = {}
  for k,v in compound_map.items():
    R_COMPOUND_MAP[v] = k

  for idx in few_shot_indexes:
    prompt += f"\n"
    prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
    prompt += f"Exurbanta: {fewshot_label_sentences[idx-1][1]}\n"

  prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
  return prompt
```

```
[ ]: def EXPERIMENT_CDBA_kitchensink_prompting(input_file, label_files,␣
     ↪fewshot_input_file, fewshot_label_files, translate_map, compound_map,␣
     ↪noising_map, log_dir, result_dir, log_fname = "", result_fname = ""):
       ### Start of Explanation
       # This code is used to perform fewshot prompting experiment
       # ONLY FOR EXPERIMENT WITH CODE 'AB'
       ### End of Explanation

       # Start of Experiment Preparation
       import os
       from tqdm import tqdm
```

```
os.makedirs(log_dir, exist_ok=True)
os.makedirs(result_dir, exist_ok=True)

## Get Experiment Details
EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = CDBA_kitchensink_prompting(input_sentence = input_sentence,
                                        label_sentence = label_sentences[idx],
                                        fewshot_input_file =␣
↪fewshot_input_file,
                                        fewshot_label_file =␣
↪fewshot_label_file,
                                        translate_map = translate_map,
                                        compound_map = compound_map,
                                        noising_map = noising_map,
                                        word_order = EXPERIMENT_order)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = 'test'
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
    # temp_log += f"{result}\n"
    temp_log += f"=====\n"

    temp_out = f"$$$ Entry {idx}\n"
    temp_out += f"{result}\n"
    temp_out += f"--ENDOFENTRY--\n"


    if log_fname == "":
      with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
        f.write(temp_log)
    else:
      with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
        f.write(temp_log)

    if result_fname == "":
      with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
        f.write(temp_out)
    else:
      with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
        f.write(temp_out)
```

D1

```
[ ]: import os

     WORK_DIR = "/content/extract"

     INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
     FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')
```

```
LABEL_FILES = [
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_sov'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_svo'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vos'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vso')
]

TRANSLATE_MAP = load_map(os.path.join(WORK_DIR, 'D1_mapping.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'D1B/D1B_compound_map.pickle'))
NOISING_MAP = load_map(os.path.join(WORK_DIR, 'D1BA/D1BA_noising_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

[ ]:
```
EXPERIMENT_CDBA_kitchensink_prompting(input_file = INPUT_FILE,
                                  label_files = LABEL_FILES,
                                  fewshot_input_file = FEWSHOT_INPUT_FILE,
                                  fewshot_label_files = FEWSHOT_LABEL_FILES,
                                  translate_map = TRANSLATE_MAP,
                                  compound_map = COMPOUND_MAP,
                                  noising_map = NOISING_MAP,
                                  log_dir = LOG_DIR,
                                  result_dir = RESULT_DIR,
                                  log_fname = "kitchensink",
                                  result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/fewshot_prompting
res_dir = experiment_results/D1BA/fewshot_prompting

100%|      | 1012/1012 [00:14<00:00, 69.02it/s]
100%|      | 1012/1012 [00:12<00:00, 77.95it/s]
100%|      | 1012/1012 [00:13<00:00, 77.17it/s]
100%|      | 1012/1012 [00:14<00:00, 70.78it/s]
```

[ ]:
```

```

3 Prompts v2

[ ]:
```
import shutil
shutil.rmtree('experiment_prompts')
shutil.rmtree('experiment_results')
```

3.1 1. Fewshot Prompting, but only give 5 random examples (Follows dito's pattern)

3.1.1 AB

[ ]:
```
def tokenize(text):
  import re
  import string
  ret = []
  for token in text.split(' '):
    result_list = re.findall(r'\w+|[^\w\s]', token)
    flag = 0
    for token in result_list:
      if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

def get_index(in_token, few_shots):
  for idx, tokens in few_shots:
    if in_token in tokens:
      return idx
  return in_token

import random
random.seed(2023)
def AB_fiveshot_prompting(input_sentence, fewshot_input_file,␣
 ↪fewshot_label_file, noising_map, word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_AB_fewshot_prompting
  # Uses the tokenize function above
  # This functiosn only require the input_sentence
  # AB.1 and AB.2 had to use label_sentence because of the existance of␣
 ↪compounding words
  ### End of Explanation
  fewshot_input_sentences = []
  with open(fewshot_input_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_input_sentences.append((i, line.strip()))
      i += 1

  fewshot_label_sentences = []
  with open(fewshot_label_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_label_sentences.append((i, line.strip()))
      i += 1


  few_shot_indexes = random.sample(range(1, 998), 5)

  prompt = "This is an English to Exurbanta translation, please provide the␣
↪Exurbanta translation for these sentences:\n"
  for idx in few_shot_indexes:
    prompt += f"English: {fewshot_input_sentences[idx-1][1].strip()} Exurbanta:␣
↪{fewshot_label_sentences[idx-1][1].strip()}\n"
  prompt += "Please provide the translation for the following sentence.\n"
  prompt += "Do not provide any explanations or text apart from the translation.
↪\n"
  prompt += f"English: {input_sentence.strip()}\n"
  prompt += "Exurbanta: "
  return prompt
```

```
[ ]: def EXPERIMENT_AB_fiveshot_prompting(input_file, label_files,␣
↪fewshot_input_file, fewshot_label_files, noising_map, log_dir, result_dir,␣
↪log_fname = "", result_fname = ""):
  ### Start of Explanation
  # This code is used to perform fewshot prompting experiment
  # ONLY FOR EXPERIMENT WITH CODE 'AB'
  ### End of Explanation

  # Start of Experiment Preparation
  import os
  from tqdm import tqdm
  os.makedirs(log_dir, exist_ok=True)
  os.makedirs(result_dir, exist_ok=True)

  ## Get Experiment Details
  EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
  EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
  EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)
```

```
os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = AB_fiveshot_prompting(input_sentence, fewshot_input_file,␣
↪fewshot_label_file, noising_map, EXPERIMENT_order)
    dialog = [
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
```

```
      # temp_log += f"{result}\n"
      temp_log += f"=====\n"

      temp_out = f"$$$ Entry {idx}\n"
      temp_out += f"{result}\n"
      temp_out += f"--ENDOFENTRY--\n"


      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

[ ]:
```
import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vso')
]

NOISING_MAP = load_map(os.path.join(WORK_DIR, 'AB/AB_noising_map.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'B/B_compound_map.pickle'))
```

```
EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_AB_fiveshot_prompting(
         INPUT_FILE,
         LABEL_FILES,
         FEWSHOT_INPUT_FILE,
         FEWSHOT_LABEL_FILES,
         NOISING_MAP,
         LOG_DIR,
         RESULT_DIR,
         log_fname = "template_fiveshot",
         result_fname = ""
     )
```

```
name = AB
log_dir = experiment_prompts/AB/fewshot_prompting
res_dir = experiment_results/AB/fewshot_prompting
```

```
100%|      | 1012/1012 [00:01<00:00, 555.31it/s]
100%|      | 1012/1012 [00:02<00:00, 352.07it/s]
100%|      | 1012/1012 [00:01<00:00, 511.08it/s]
100%|      | 1012/1012 [00:01<00:00, 565.72it/s]
```

3.1.2 C/DxBA

```
[ ]: def tokenize(text):
       import re
       import string
       ret = []
       for token in text.split(' '):
         result_list = re.findall(r'\w+|[^\w\s]', token)
         flag = 0
         for token in result_list:
           if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
             flag = 1
         if flag:
           ret.append(''.join(result_list))
         elif len(result_list) > 2:
           ret.append(''.join(result_list))
         else:
           ret.extend(result_list)
       return ret
```

```
def get_index(in_token, few_shots):
  for idx, tokens in few_shots:
    if in_token in tokens:
      return idx
  return in_token

import random
random.seed(2023)
def CDBA_fiveshot_prompting(input_sentence, label_sentence, fewshot_input_file,␣
 ↪fewshot_label_file, translation_map, compound_map, noising_map, word_order):
  ### Start of Explanation
  # Helper function for EXPERIMENT_CDBA_fewshot_prompting
  # Uses the tokenize function above
  # This functiosn only require the input_sentence
  # Other prompting functions had to use label_sentence because of the␣
 ↪existance of compounding words
  ### End of Explanation
  fewshot_input_sentences = []
  with open(fewshot_input_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_input_sentences.append((i, line.strip()))
      i += 1

  fewshot_label_sentences = []
  with open(fewshot_label_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_label_sentences.append((i, line.strip()))
      i += 1

  few_shot_indexes = random.sample(range(1, 998), 5)

  prompt = "This is an English to Exurbanta translation, please provide the␣
 ↪Exurbanta translation for these sentences:\n"
  for idx in few_shot_indexes:
    prompt += f"English: {fewshot_input_sentences[idx-1][1].strip()} Exurbanta:␣
 ↪{fewshot_label_sentences[idx-1][1].strip()}\n"
  prompt += "Please provide the translation for the following sentence.\n"
  prompt += "Do not provide any explanations or text apart from the translation.
 ↪\n"
  prompt += f"English: {input_sentence.strip()}\n"
  prompt += "Exurbanta: "
  return prompt
```

```
[ ]: def EXPERIMENT_CDBA_fiveshot_prompting(input_file, label_files,␣
     ↪fewshot_input_file, fewshot_label_files, translate_map, compound_map,␣
     ↪noising_map, log_dir, result_dir, log_fname = "", result_fname = ""):
       ### Start of Explanation
       # This code is used to perform fewshot prompting experiment
       # ONLY FOR EXPERIMENT WITH CODE 'AB'
       ### End of Explanation

       # Start of Experiment Preparation
       import os
       from tqdm import tqdm
       os.makedirs(log_dir, exist_ok=True)
       os.makedirs(result_dir, exist_ok=True)

       ## Get Experiment Details
       EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
       EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
       EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

       os.makedirs(EXPERIMENT_logdir, exist_ok=True)
       os.makedirs(EXPERIMENT_resdir, exist_ok=True)

       EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
       EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

       os.makedirs(EXPERIMENT_logdir, exist_ok=True)
       os.makedirs(EXPERIMENT_resdir, exist_ok=True)

       ### Start of Debug
       print(f"name = {EXPERIMENT_name}")
       print(f"log_dir = {EXPERIMENT_logdir}")
       print(f"res_dir = {EXPERIMENT_resdir}")
       ### End of Debug

       input_sentences = load_file(input_file)
       for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):

         ## Create folder preparations
         EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
         EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
         EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
         os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
         os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
         ## End of Experiment Preparation

         # Start Experiment
         label_sentences = load_file(label_file)
```

```
    assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

    prompts = []
    for idx, input_sentence in enumerate(tqdm(input_sentences)):
      prompt = CDBA_fiveshot_prompting(input_sentence,
                                       label_sentences[idx],
                                       fewshot_input_file,
                                       fewshot_label_file,
                                       translate_map,
                                       compound_map,
                                       noising_map,
                                       EXPERIMENT_order)
      dialog = [
          {'role': 'user', 'content': prompt}
      ]
      payload = {
        "inputs": [dialog],
        "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
      }
      # result = query_endpoint(payload)[0]['generation']['content']
      result = 'test'
      # temp_log = f"$$$ Entry {idx}\n"
      temp_log = f"{prompt}\n"
      # temp_log += f"{result}\n"
      temp_log += f"=====\n"

      temp_out = f"$$$ Entry {idx}\n"
      temp_out += f"{result}\n"
      temp_out += f"--ENDOFENTRY--\n"


      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

D1

```
[ ]: import os

     WORK_DIR = "/content/extract"

     INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
     FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

     LABEL_FILES = [
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_sov'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_svo'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vos'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_vso')
     ]

     FEWSHOT_LABEL_FILES = [
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_sov'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_svo'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vos'),
         os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_vso')
     ]

     TRANSLATE_MAP = load_map(os.path.join(WORK_DIR, 'D1_mapping.pickle'))
     COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'D1B/D1B_compound_map.pickle'))
     NOISING_MAP = load_map(os.path.join(WORK_DIR, 'D1BA/D1BA_noising_map.pickle'))

     LOG_DIR = "experiment_prompts/"
     RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_CDBA_fiveshot_prompting(input_file = INPUT_FILE,
                                        label_files = LABEL_FILES,
                                        fewshot_input_file = FEWSHOT_INPUT_FILE,
                                        fewshot_label_files = FEWSHOT_LABEL_FILES,
                                        translate_map = TRANSLATE_MAP,
                                        compound_map = COMPOUND_MAP,
                                        noising_map = NOISING_MAP,
                                        log_dir = LOG_DIR,
                                        result_dir = RESULT_DIR,
                                        log_fname = "template_fiveshot-test1",
                                        result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/fewshot_prompting
res_dir = experiment_results/D1BA/fewshot_prompting

100%|      | 1012/1012 [00:01<00:00, 582.91it/s]
100%|      | 1012/1012 [00:01<00:00, 581.99it/s]
100%|      | 1012/1012 [00:01<00:00, 595.37it/s]
```

```
100%|          | 1012/1012 [00:01<00:00, 511.19it/s]
```

[ ]:

```

```

3.2 2. Bilingual Mapping with Word Type 50% Masked

–> ONLY RUN THIS FOR SVO

[ ]:

```
# EXTRACT

import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_english_vso')
]

FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_sov'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_svo'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vos'),
    os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_vso')
]

NOISING_MAP = load_map(os.path.join(WORK_DIR, 'AB/AB_noising_map.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'B/B_compound_map.pickle'))

EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

[ ]:

```
# import pandas as pd
# import nltk
# nltk.download('punkt')
# nltk.download('universal_tagset')
# nltk.download('averaged_perceptron_tagger')
# import nltk
# from nltk.tokenize import word_tokenize, sent_tokenize
# from nltk.tag import pos_tag

# sentences = []
# with open('extract/flores_english_svo', 'r', encoding="utf-8") as f:
#   for line in f:
#     sentences.append(line)

# # Tokenize each sentence into words and tag with POS
# word_counter = {}  # Set to store unique word types
# for sentence in sentences:
#     words = word_tokenize(sentence)
#     tagged_words = pos_tag(words, tagset="universal")

#     # Extract the POS tags and add to the set
#     words = [word for word, tag in tagged_words if tag not in ["NOUN",␣
 ↪"VERB", "ADJ"]]
#     for word in words:
#       if word in word_counter:
#         word_counter[word] += 1
#       else:
#         word_counter[word] = 1

# df = pd.DataFrame(list(word_counter.items()), columns=["Word", "Count"])
# df = df.sort_values(by="Count", ascending=False)

# df.to_excel('all_OTHER_universal.xlsx', index=False)

# half_len = len(df) // 2
# df = df.head(half_len)
# df.to_excel('allowed_OTHER.xlsx', index=False)
```

3.2.1 AB

Algorithms: 1. Load the original corpus 2. Keep track of all the Noun words and how much they occur. 3. Remove 50% of the least occuring Nouns, turn the keys to list. 4. Perform prompting like usual, BUT, if it is a Noun, check if the word is in the ALLOWED_NOUN. If not, don't give the translation.

Uses: 1. allowed_adj.xlsx 2. allowed_noun.xlsx 3. allowed_verb.xlsx 4. allowed_OTHER.xlsx

[ ]:
```
def load_allowed(path):
  import pandas as pd

  df = pd.read_excel(path)
  word_list = df['Word'].astype(str).tolist()
  return word_list
```

```
[ ]: import nltk
nltk.download('punkt')
nltk.download('universal_tagset')

import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tag import pos_tag

def tokenize(text):
  import re
  import string
  ret = []
  for token in text.split(' '):
    result_list = re.findall(r'\w+|[^\w\s]', token)
    flag = 0
    for token in result_list:
      if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

def AB_bilingual_prompting(input_sentence, label_sentence, noising_map,␣
 ↪compound_map, word_order, allowed_list, word_type):
  ### Start of Explanation
  # Helper function for EXPERIMENT_AB_bilingual_prompting
  # Uses the tokenize function above
  ### End of Explanation

  assert word_type in ["ADJ", "NOUN", "VERB", "OTHER"], print(f"{word_type} not␣
 ↪in allowed word type")

  LABEL_TOKENS = tokenize(label_sentence)
  R_NOISING_MAP = {}
  for k,v in noising_map.items():
    R_NOISING_MAP[v] = k

  R_COMPOUND_MAP = {}
  for k,v in compound_map.items():
    R_COMPOUND_MAP[v] = k
```

```
NOISED_TOKENS = []
COMPOUNDED_TOKENS = []

# word, tag = pos_tag([token], tagset="universal")[0]
# if tag == word_type:
#   if word not in allowed_list:
#     continue

for token in LABEL_TOKENS:
  if token in R_COMPOUND_MAP.keys():
    COMPOUNDED_TOKENS.append(token)

  if token in R_NOISING_MAP.keys():
    NOISED_TOKENS.append(token)

prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
if word_order == "sov":
  prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
elif word_order == "svo":
  prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
elif word_order == "vos":
  prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
elif word_order == "vso":
  prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
prompt += "The following is a list of word translations from English to␣
↪Exurbanta:\n"

ALREADY_TRANSLATED = []

for token in COMPOUNDED_TOKENS:
  if token not in ALREADY_TRANSLATED:
    ALREADY_TRANSLATED.append(token)
    prompt += f'"{R_COMPOUND_MAP[token][0]} {R_COMPOUND_MAP[token][1]}" means␣
↪"{token}"\n'

for token in NOISED_TOKENS:
  word, tag = pos_tag([R_NOISING_MAP[token]], tagset="universal")[0]
  if tag == word_type:
    if word not in allowed_list:
      continue
  if token not in ALREADY_TRANSLATED:
    ALREADY_TRANSLATED.append(token)
    prompt += f'"{R_NOISING_MAP[token]}" means "{token}"\n'

prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
```

```
    return prompt
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package universal_tagset to /root/nltk_data…
[nltk_data]   Package universal_tagset is already up-to-date!
```

[ ]:
```
def EXPERIMENT_AB_bilingual_prompting(input_file, label_files, noising_map,␣
 ↪compound_map, allowed_list, word_type, log_dir, result_dir, log_fname = "",␣
 ↪result_fname = ""):
  ### Start of Explanation
  # This code is used to perform bilingual prompting experiment
  # ONLY FOR EXPERIMENT WITH CODE 'AB'
  ### End of Explanation

  # Start of Experiment Preparation
  import os
  from tqdm import tqdm
  os.makedirs(log_dir, exist_ok=True)
  os.makedirs(result_dir, exist_ok=True)

  ## Get Experiment Details
  EXPERIMENT_name  = label_files[0].split('/')[-1].split('_')[0]
  EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
  EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

  os.makedirs(EXPERIMENT_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_resdir, exist_ok=True)

  EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'bilingual_prompting')
  EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'bilingual_prompting')

  os.makedirs(EXPERIMENT_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_resdir, exist_ok=True)

  ### Start of Debug
  print(f"name = {EXPERIMENT_name}")
  print(f"log_dir = {EXPERIMENT_logdir}")
  print(f"res_dir = {EXPERIMENT_resdir}")
  ### End of Debug

  input_sentences = load_file(input_file)
  for label_file in label_files:

    ## Create folder preparations
    EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
    EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = AB_bilingual_prompting(input_sentence, label_sentences[idx],␣
↪noising_map, compound_map, EXPERIMENT_order, allowed_list, word_type)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
    # temp_log += f"{result}\n"
    temp_log += f"=====\n"

    temp_out = f"$$$ Entry {idx}\n"
    temp_out += f"{result}\n"
    temp_out += f"--ENDOFENTRY--\n"


    if log_fname == "":
      with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
        f.write(temp_log)
    else:
      with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
        f.write(temp_log)

    if result_fname == "":
      with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
        f.write(temp_out)
    else:
      with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
        f.write(temp_out)
```

ADJ

```
[ ]: import os

     WORK_DIR = "/content/extract"

     INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
     FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

     LABEL_FILES = [
         os.path.join(WORK_DIR, 'AB/AB_flores_english_svo')
     ]

     FEWSHOT_LABEL_FILES = [
         os.path.join(WORK_DIR, 'AB/AB_flores_dev_english_svo')
     ]

     NOISING_MAP = load_map(os.path.join(WORK_DIR, 'AB/AB_noising_map.pickle'))
     COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'B/B_compound_map.pickle'))

     EN_DE_MAP = load_map(os.path.join(WORK_DIR, 'en_de_map.pickle'))
     EN_PT_MAP = load_map(os.path.join(WORK_DIR, 'en_pt_map.pickle'))
     EN_AF_MAP = load_map(os.path.join(WORK_DIR, 'en_af_map.pickle'))
     EN_GL_MAP = load_map(os.path.join(WORK_DIR, 'en_gl_map.pickle'))

     LOG_DIR = "experiment_prompts/"
     RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_AB_bilingual_prompting(input_file = INPUT_FILE,
                                       label_files = LABEL_FILES,
                                       noising_map = NOISING_MAP,
                                       compound_map = COMPOUND_MAP,
                                       allowed_list = load_allowed('allowed_adj.
       ↪xlsx'),
                                       word_type = "ADJ",
                                       log_dir = LOG_DIR,
                                       result_dir = RESULT_DIR,
                                       log_fname = "masked_adj",
                                       result_fname = "")
```

```
name = AB
log_dir = experiment_prompts/AB/bilingual_prompting
res_dir = experiment_results/AB/bilingual_prompting

100%|          | 1012/1012 [00:08<00:00, 124.69it/s]
```

NOUN

```
[ ]: EXPERIMENT_AB_bilingual_prompting(input_file = INPUT_FILE,
                                  label_files = LABEL_FILES,
                                  noising_map = NOISING_MAP,
                                  compound_map = COMPOUND_MAP,
                                  allowed_list = load_allowed('allowed_noun.
 ↪xlsx'),
                                  word_type = "NOUN",
                                  log_dir = LOG_DIR,
                                  result_dir = RESULT_DIR,
                                  log_fname = "masked_noun",
                                  result_fname = "")
```

```
name = AB
log_dir = experiment_prompts/AB/bilingual_prompting
res_dir = experiment_results/AB/bilingual_prompting
```

```
100%|      | 1012/1012 [00:08<00:00, 116.78it/s]
```

VERB

```
[ ]: EXPERIMENT_AB_bilingual_prompting(input_file = INPUT_FILE,
                                  label_files = LABEL_FILES,
                                  noising_map = NOISING_MAP,
                                  compound_map = COMPOUND_MAP,
                                  allowed_list = load_allowed('allowed_verb.
 ↪xlsx'),
                                  word_type = "VERB",
                                  log_dir = LOG_DIR,
                                  result_dir = RESULT_DIR,
                                  log_fname = "masked_verb",
                                  result_fname = "")
```

```
name = AB
log_dir = experiment_prompts/AB/bilingual_prompting
res_dir = experiment_results/AB/bilingual_prompting
```

```
100%|      | 1012/1012 [00:07<00:00, 142.06it/s]
```

OTHER

```
[ ]: EXPERIMENT_AB_bilingual_prompting(input_file = INPUT_FILE,
                                  label_files = LABEL_FILES,
                                  noising_map = NOISING_MAP,
                                  compound_map = COMPOUND_MAP,
                                  allowed_list = load_allowed('allowed_OTHER.
 ↪xlsx'),
                                  word_type = "OTHER",
                                  log_dir = LOG_DIR,
                                  result_dir = RESULT_DIR,
                                  log_fname = "masked_other",
                                  result_fname = "")
```

```
name = AB
log_dir = experiment_prompts/AB/bilingual_prompting
res_dir = experiment_results/AB/bilingual_prompting
```

```
100%|      | 1012/1012 [00:08<00:00, 120.40it/s]
```

3.2.2 C/DxBA - D1

[ ]:
```
import nltk
nltk.download('punkt')
nltk.download('universal_tagset')

import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tag import pos_tag

def tokenize(text):
  import re
  import string
  ret = []
  for token in text.split(' '):
    result_list = re.findall(r'\w+|[^\w\s]', token)
    flag = 0
    for token in result_list:
      if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

def CDBA_bilingual_prompting(input_sentence, label_sentence, translate_map,␣
 ↪compound_map, noising_map, word_order, allowed_list, word_type):
  ### Start of Explanation
  # Helper function for EXPERIMENT_CDBA_bilingual_prompting
  # Uses the tokenize function above (tbh, they are all the same and unchanged)
  ### End of Explanation

  assert word_type in ["ADJ", "NOUN", "VERB", "OTHER"], print(f"{word_type} not␣
 ↪in allowed word type")
```

```python
# 0. Prepare prompt
prompt = "Exurbanta is a lost language to humanity that was found only a few␣
↪days ago.\n"
if word_order == "sov":
  prompt += "Exurbanta follows the Subject-Object-Verb word order.\n"
elif word_order == "svo":
  prompt += "Exurbanta follows the Subject-Verb-Object word order.\n"
elif word_order == "vos":
  prompt += "Exurbanta follows the Verb-Object-Subject word order .\n"
elif word_order == "vso":
  prompt += "Exurbanta follows the Verb-Subject-Object word order .\n"
prompt += "The following is a list of word translations from English to␣
↪Exurbanta:\n"

ALREADY_TRANSLATED = []

INPUT_TOKENS = tokenize(input_sentence)
# 1. Original --> C/Dx
for token in INPUT_TOKENS:
  word, tag = pos_tag([token], tagset="universal")[0]
  if token in translate_map:
    if token not in ALREADY_TRANSLATED:
      ALREADY_TRANSLATED.append(token)
      ALREADY_TRANSLATED.append(TRANSLATE_MAP[token])

      if tag == word_type:
        if word not in allowed_list:
          if TRANSLATE_MAP[token] in NOISING_MAP:
            ALREADY_TRANSLATED.append(NOISING_MAP[TRANSLATE_MAP[token]])
          continue

      if TRANSLATE_MAP[token] in NOISING_MAP:
        prompt += f'"{token}" means "{NOISING_MAP[TRANSLATE_MAP[token]]}"\n'
        ALREADY_TRANSLATED.append(NOISING_MAP[TRANSLATE_MAP[token]])
      else:
        prompt += f'"{token}" means "{TRANSLATE_MAP[token]}"\n'


# 2. C/Dx --> C/DxB --> C/DxBA
## Note, Even though the A + B experiment is named AB,
## It actually perform compounding first, THEN noising
## Just like this one.
LABEL_TOKENS = tokenize(label_sentence)

R_COMPOUND_MAP = {}
for k,v in compound_map.items():
  R_COMPOUND_MAP[v] = k
```

```
R_NOISING_MAP = {}
for k,v in noising_map.items():
  R_NOISING_MAP[v] = k

COMPOUNDED_TOKENS = []
NOISED_TOKENS = []

for token in LABEL_TOKENS:
  if token in R_NOISING_MAP.keys():
    NOISED_TOKENS.append(token)
    if token in R_COMPOUND_MAP.keys():
      COMPOUNDED_TOKENS.append(token)

for token in COMPOUNDED_TOKENS:
  if token not in ALREADY_TRANSLATED:
    ALREADY_TRANSLATED.append(token)
    prompt += f'"{R_COMPOUND_MAP[token][0]} {R_COMPOUND_MAP[token][1]}" means␣
↪"{token}"\n'

for token in NOISED_TOKENS:
  word, tag = pos_tag([R_NOISING_MAP[token]], tagset="universal")[0]
  if tag == word_type:
    if word not in allowed_list:
      continue

  if token not in ALREADY_TRANSLATED:
    ALREADY_TRANSLATED.append(token)
    prompt += f'"{R_NOISING_MAP[token]}" means "{token}"\n'

prompt += f'Translate the following text from English into Exurbanta:
↪\n{input_sentence}'
return prompt
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package universal_tagset to /root/nltk_data…
[nltk_data]   Package universal_tagset is already up-to-date!
```

```
[ ]: def EXPERIMENT_CDBA_bilingual_prompting(input_file, label_files, translate_map,␣
↪compound_map, noising_map, allowed_list, word_type, log_dir, result_dir,␣
↪log_fname = "", result_fname = ""):
  ### Start of Explanation
  # This code is used to perform bilingual prompting experiment
  # ONLY FOR EXPERIMENT WITH CODE 'CDBA'
  ### End of Explanation
```

```
# Start of Experiment Preparation
import os
from tqdm import tqdm
os.makedirs(log_dir, exist_ok=True)
os.makedirs(result_dir, exist_ok=True)

## Get Experiment Details
EXPERIMENT_name   = label_files[0].split('/')[-1].split('_')[0]
EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'bilingual_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'bilingual_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file in label_files:

  ## Create folder preparations
  EXPERIMENT_order = label_file.split('/')[-1].split('_')[-1]
  EXPERIMENT_order_logdir = os.path.join(EXPERIMENT_logdir, EXPERIMENT_order)
  EXPERIMENT_order_resdir = os.path.join(EXPERIMENT_resdir, EXPERIMENT_order)
  os.makedirs(EXPERIMENT_order_logdir, exist_ok=True)
  os.makedirs(EXPERIMENT_order_resdir, exist_ok=True)
  ## End of Experiment Preparation

  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = CDBA_bilingual_prompting(input_sentence, label_sentences[idx],␣
↪translate_map, compound_map, noising_map, EXPERIMENT_order, allowed_list,␣
↪word_type)
```

```
      dialog = [
          {'role': 'system', 'content': 'You can only use one sentence.'},
          {'role': 'user', 'content': prompt}
      ]
      payload = {
        "inputs": [dialog],
        "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
      }
      # result = query_endpoint(payload)[0]['generation']['content']
      result = "test"
      # temp_log = f"$$$ Entry {idx}\n"
      temp_log = f"{prompt}\n"
      # temp_log += f"{result}\n"
      temp_log += f"=====\n"

      temp_out = f"$$$ Entry {idx}\n"
      temp_out += f"{result}\n"
      temp_out += f"--ENDOFENTRY--\n"

      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
  ↪f:
          f.write(temp_out)
```

ADJ

[ ]:
```
import os

WORK_DIR = "/content/extract"

INPUT_FILE = os.path.join(WORK_DIR, 'flores_english_svo')
FEWSHOT_INPUT_FILE = os.path.join(WORK_DIR, 'flores_dev_english_svo')

LABEL_FILES = [
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_english_svo')
]
```

```
FEWSHOT_LABEL_FILES = [
    os.path.join(WORK_DIR, 'D1BA/D1BA_flores_dev_english_svo')
]

TRANSLATE_MAP = load_map(os.path.join(WORK_DIR, 'D1_mapping.pickle'))
COMPOUND_MAP = load_map(os.path.join(WORK_DIR, 'D1B/D1B_compound_map.pickle'))
NOISING_MAP = load_map(os.path.join(WORK_DIR, 'D1BA/D1BA_noising_map.pickle'))

LOG_DIR = "experiment_prompts/"
RESULT_DIR = "experiment_results/"
```

```
[ ]: EXPERIMENT_CDBA_bilingual_prompting(input_file = INPUT_FILE,
                                    label_files = LABEL_FILES,
                                    translate_map = TRANSLATE_MAP,
                                    compound_map = COMPOUND_MAP,
                                    noising_map = NOISING_MAP,
                                    allowed_list = load_allowed('allowed_adj.
      ↪xlsx'),
                                    word_type = "ADJ",
                                    log_dir = LOG_DIR,
                                    result_dir = RESULT_DIR,
                                    log_fname = "masked_adj",
                                    result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/bilingual_prompting
res_dir = experiment_results/D1BA/bilingual_prompting

100%|      | 1012/1012 [00:14<00:00, 68.56it/s]
```

NOUN

```
[ ]: EXPERIMENT_CDBA_bilingual_prompting(input_file = INPUT_FILE,
                                    label_files = LABEL_FILES,
                                    translate_map = TRANSLATE_MAP,
                                    compound_map = COMPOUND_MAP,
                                    noising_map = NOISING_MAP,
                                    allowed_list = load_allowed('allowed_noun.
      ↪xlsx'),
                                    word_type = "NOUN",
                                    log_dir = LOG_DIR,
                                    result_dir = RESULT_DIR,
                                    log_fname = "masked_noun",
                                    result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/bilingual_prompting
res_dir = experiment_results/D1BA/bilingual_prompting
```

```
100%|     | 1012/1012 [00:14<00:00, 70.54it/s]
```

VERB

```
[ ]: EXPERIMENT_CDBA_bilingual_prompting(input_file = INPUT_FILE,
                                        label_files = LABEL_FILES,
                                        translate_map = TRANSLATE_MAP,
                                        compound_map = COMPOUND_MAP,
                                        noising_map = NOISING_MAP,
                                        allowed_list = load_allowed('allowed_verb.
      ↪xlsx'),
                                        word_type = "VERB",
                                        log_dir = LOG_DIR,
                                        result_dir = RESULT_DIR,
                                        log_fname = "masked_verb",
                                        result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/bilingual_prompting
res_dir = experiment_results/D1BA/bilingual_prompting
```

```
100%|     | 1012/1012 [00:14<00:00, 70.38it/s]
```

OTHER

```
[ ]: EXPERIMENT_CDBA_bilingual_prompting(input_file = INPUT_FILE,
                                        label_files = LABEL_FILES,
                                        translate_map = TRANSLATE_MAP,
                                        compound_map = COMPOUND_MAP,
                                        noising_map = NOISING_MAP,
                                        allowed_list = load_allowed('allowed_OTHER.
      ↪xlsx'),
                                        word_type = "OTHER",
                                        log_dir = LOG_DIR,
                                        result_dir = RESULT_DIR,
                                        log_fname = "masked_other",
                                        result_fname = "")
```

```
name = D1BA
log_dir = experiment_prompts/D1BA/bilingual_prompting
res_dir = experiment_results/D1BA/bilingual_prompting
```

```
100%|     | 1012/1012 [00:12<00:00, 78.90it/s]
```

4 Prompts v3

English to X (Code: EX)

4.1 EX

4.1.1 Fewshot

```
[ ]: ## Helper func
def load_map(f_name):
  import pickle
  with open(f_name, 'rb') as f:
    return pickle.load(f)

def load_file(f_name):
  print(f"attempting to open {f_name}")
  with open(f_name, 'r') as f:
    return f.readlines()
## End of helper func
```

```
[ ]: def tokenize(text):
  import re
  import string
  ret = []
  for token in text.split(' '):
    result_list = re.findall(r'\w+|[^\w\s]', token)
    flag = 0
    for token in result_list:
      if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
        flag = 1
    if flag:
      ret.append(''.join(result_list))
    elif len(result_list) > 2:
      ret.append(''.join(result_list))
    else:
      ret.extend(result_list)
  return ret

def get_index(in_token, few_shots):
  for idx, tokens in few_shots:
    if in_token in tokens:
      return idx
  return in_token

import random
random.seed(2023)
def EX_fewshot_prompting(input_sentence, fewshot_input_file,␣
 ↪fewshot_label_file, translation_map, language):
  fewshot_input_sentences = []
  with open(fewshot_input_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_input_sentences.append((i, line.strip()))
      i += 1

  fewshot_label_sentences = []
  with open(fewshot_label_file, 'r') as f:
    i = 1
    for line in f:
      fewshot_label_sentences.append((i, line.strip()))
      i += 1

  input_tokens = tokenize(input_sentence)
  few_shot_indexes = []
  unfound_word = []
  for input_token in input_tokens:
    idx = get_index(input_token, fewshot_input_sentences)
    if isinstance(idx, int):
      few_shot_indexes.append(get_index(input_token, fewshot_input_sentences))
    elif isinstance(idx, str):
      unfound_word.append(idx)

  few_shot_indexes = list(set(few_shot_indexes))
  unfound_word = list(set(unfound_word)) # Not handled

  prompt = f"The following is a list of sentence translations from English to␣
↪{language}:\n"
  for idx in few_shot_indexes:
    prompt += f"English: {fewshot_input_sentences[idx-1][1]}\n"
    prompt += f"{language}: {fewshot_label_sentences[idx-1][1]}\n"

  for unfound_en_word in unfound_word:
    prompt += f"English: {unfound_en_word}\n"
    prompt += f"{language}: {translation_map.get(unfound_en_word,␣
↪unfound_en_word)}\n"

  prompt += f'Translate the following text from English into {language}:
↪\n{input_sentence}'
  return prompt
```

```
[ ]: def EXPERIMENT_EX_fewshot_prompting(input_file, label_files,␣
      ↪fewshot_input_file, fewshot_label_files, translation_map, language, log_dir,␣
      ↪result_dir, log_fname = "", result_fname = ""):
       ### Start of Explanation
       # This code is used to perform fewshot prompting experiment
       # ONLY FOR EXPERIMENT WITH CODE 'AB'
       ### End of Explanation

       # Start of Experiment Preparation
```

```python
import os
from tqdm import tqdm
os.makedirs(log_dir, exist_ok=True)
os.makedirs(result_dir, exist_ok=True)

## Get Experiment Details
EXPERIMENT_name  = "ES"
EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):
  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = EX_fewshot_prompting(input_sentence, fewshot_input_file,␣
↪fewshot_label_file, translation_map, language)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
```

```
      temp_log = f"{prompt}\n"
      # temp_log += f"{result}\n"
      temp_log += f"=====\n"

      temp_out = f"$$$ Entry {idx}\n"
      temp_out += f"{result}\n"
      temp_out += f"--ENDOFENTRY--\n"

      EXPERIMENT_order_logdir = os.getcwd()
      EXPERIMENT_order_resdir = os.getcwd()
      if log_fname == "":
        with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
          f.write(temp_log)
      else:
        with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
          f.write(temp_log)

      if result_fname == "":
        with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
          f.write(temp_out)
      else:
        with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
          f.write(temp_out)
```

```
[ ]: EXPERIMENT_EX_fewshot_prompting(input_file = "/content/eng_Latn.devtest",
                                     label_files = ["/content/eus_Latn.devtest"],
                                     fewshot_input_file = "/content/eng_Latn.dev",
                                     fewshot_label_files = ["/content/eus_Latn.dev"],
                                     translation_map = load_map("/content/EB_mapping.
↪pickle"),
                                     language = "Basque"
                                     log_dir = "Prompts",
                                     result_dir = " Results",
                                     log_fname = "EB_prompt_fewshot",
                                     result_fname = "")
```

```
name = ES
log_dir = Prompts/ES/fewshot_prompting
res_dir =  Results/ES/fewshot_prompting
attempting to open /content/eng_Latn.devtest
attempting to open /content/eus_Latn.devtest

100%|     | 1012/1012 [00:04<00:00, 203.87it/s]
```

4.1.2 Bilingual

```
[ ]: def tokenize(text):
       import re
       import string
       ret = []
       for token in text.split(' '):
         result_list = re.findall(r'\w+|[^\w\s]', token)
         flag = 0
         for token in result_list:
           if token in ['(', '[', '{', '}', ']', ')', '"', "'"]:
             flag = 1
         if flag:
           ret.append(''.join(result_list))
         elif len(result_list) > 2:
           ret.append(''.join(result_list))
         else:
           ret.extend(result_list)
       return ret

     def EX_bilingual_prompting(input_sentence, translation_map, language):
       prompt = f"The following is a list of word translations from English to␣
      ↪{language}:\n"
       input_tokens = tokenize(input_sentence)

       ALREADY_TRANSLATED = []
       for input_token in input_tokens:
         if input_token not in ALREADY_TRANSLATED:
           ALREADY_TRANSLATED.append(input_token)
           prompt += f'"{input_token}" means "{translation_map.get(input_token,␣
      ↪input_token)}"\n'

       prompt += f'Translate the following text from English into {language}:
      ↪\n{input_sentence}'
       return prompt
```

```
[ ]: def EXPERIMENT_EX_bilingual_prompting(input_file, label_files,␣
      ↪fewshot_input_file, fewshot_label_files, translation_map, language, log_dir,␣
      ↪result_dir, log_fname = "", result_fname = ""):
       ### Start of Explanation
       # This code is used to perform fewshot prompting experiment
       # ONLY FOR EXPERIMENT WITH CODE 'AB'
       ### End of Explanation

       # Start of Experiment Preparation
       import os
       from tqdm import tqdm
```

```
os.makedirs(log_dir, exist_ok=True)
os.makedirs(result_dir, exist_ok=True)

## Get Experiment Details
EXPERIMENT_name  = "ES"
EXPERIMENT_logdir = os.path.join(log_dir, EXPERIMENT_name)
EXPERIMENT_resdir = os.path.join(result_dir, EXPERIMENT_name)

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

EXPERIMENT_logdir = os.path.join(EXPERIMENT_logdir, 'fewshot_prompting')
EXPERIMENT_resdir = os.path.join(EXPERIMENT_resdir, 'fewshot_prompting')

os.makedirs(EXPERIMENT_logdir, exist_ok=True)
os.makedirs(EXPERIMENT_resdir, exist_ok=True)

### Start of Debug
print(f"name = {EXPERIMENT_name}")
print(f"log_dir = {EXPERIMENT_logdir}")
print(f"res_dir = {EXPERIMENT_resdir}")
### End of Debug

input_sentences = load_file(input_file)
for label_file, fewshot_label_file in zip(label_files, fewshot_label_files):
  # Start Experiment
  label_sentences = load_file(label_file)
  assert len(input_sentences) == len(label_sentences), print("FILE LENGTH␣
↪DONT MATCH")

  prompts = []
  for idx, input_sentence in enumerate(tqdm(input_sentences)):
    prompt = EX_bilingual_prompting(input_sentence, translation_map, language)
    dialog = [
        {'role': 'system', 'content': 'You can only use one sentence.'},
        {'role': 'user', 'content': prompt}
    ]
    payload = {
      "inputs": [dialog],
      "parameters": {"max_new_tokens": 384, "top_p": 0.9, "temperature": 0.01}
    }
    # result = query_endpoint(payload)[0]['generation']['content']
    result = "test"
    # temp_log = f"$$$ Entry {idx}\n"
    temp_log = f"{prompt}\n"
    # temp_log += f"{result}\n"
    temp_log += f"=====\n"
    temp_out = f"$$$ Entry {idx}\n"
    temp_out += f"{result}\n"
    temp_out += f"--ENDOFENTRY--\n"

    EXPERIMENT_order_logdir = os.getcwd()
    EXPERIMENT_order_resdir = os.getcwd()
    if log_fname == "":
      with open(os.path.join(EXPERIMENT_order_logdir, f'log'), 'a') as f:
        f.write(temp_log)
    else:
      with open(os.path.join(EXPERIMENT_order_logdir, log_fname), 'a') as f:
        f.write(temp_log)

    if result_fname == "":
      with open(os.path.join(EXPERIMENT_order_resdir, f'result'), 'a') as f:
        f.write(temp_out)
    else:
      with open(os.path.join(EXPERIMENT_order_resdir, result_fname), 'a') as␣
↪f:
        f.write(temp_out)
```

```
[ ]: EXPERIMENT_EX_bilingual_prompting(input_file = "/content/eng_Latn.devtest",
                                     label_files = ["/content/eus_Latn.devtest"],
                                     fewshot_input_file = "/content/eng_Latn.dev",
                                     fewshot_label_files = ["/content/eus_Latn.dev"],
                                     translation_map = load_map("/content/EB_mapping.
↪pickle"),
                                     language = "Basque"
                                     log_dir = "Prompts",
                                     result_dir = " Results",
                                     log_fname = "EB_prompt_bilingual",
                                     result_fname = "")
```

```
name = ES
log_dir = Prompts/ES/fewshot_prompting
res_dir =  Results/ES/fewshot_prompting
attempting to open /content/eng_Latn.devtest
attempting to open /content/eus_Latn.devtest

100%|      | 1012/1012 [00:00<00:00, 4286.73it/s]
```

4.2 English –> Afrikaans

```
[ ]: EXPERIMENT_EX_fewshot_prompting(input_file = "/content/eng_Latn.devtest",
                                     label_files = ["/content/afr_Latn.devtest"],
                                     fewshot_input_file = "/content/eng_Latn.dev",
                                     fewshot_label_files = ["/content/afr_Latn.dev"],
                                     translation_map = load_map("/content/en_af_map.
     ↪pickle"),
                                     language = "Afrikaans",
                                     log_dir = "Prompts",
                                     result_dir = " Results",
                                     log_fname = "en_af_prompt_fewshot",
                                     result_fname = "")
```

```
name = ES
log_dir = Prompts/ES/fewshot_prompting
res_dir =  Results/ES/fewshot_prompting
attempting to open /content/eng_Latn.devtest
attempting to open /content/afr_Latn.devtest

100%|      | 1012/1012 [00:03<00:00, 283.21it/s]
```

[ ]:
```
EXPERIMENT_EX_bilingual_prompting(input_file = "/content/eng_Latn.devtest",
                                label_files = ["/content/afr_Latn.devtest"],
                                fewshot_input_file = "/content/eng_Latn.dev",
                                fewshot_label_files = ["/content/afr_Latn.dev"],
                                translation_map = load_map("/content/en_af_map.
↪pickle"),
                                language = "Afrikaans",
                                log_dir = "Prompts",
                                result_dir = " Results",
                                log_fname = "en_af_prompt_bilingual",
                                result_fname = "")
```

```
name = ES
log_dir = Prompts/ES/fewshot_prompting
res_dir =  Results/ES/fewshot_prompting
attempting to open /content/eng_Latn.devtest
attempting to open /content/afr_Latn.devtest

100%|      | 1012/1012 [00:00<00:00, 4960.54it/s]
```

4.3 English –> Tamil

[ ]:
```
EXPERIMENT_EX_fewshot_prompting(input_file = "/content/eng_Latn.devtest",
                                label_files = ["/content/tam_Taml.devtest"],
                                fewshot_input_file = "/content/eng_Latn.dev",
                                fewshot_label_files = ["/content/tam_Taml.dev"],
                                translation_map = load_map("/content/
↪en_ta_mapping.pickle"),
                                language = "Tamil",
                                log_dir = "Prompts",
                                result_dir = " Results",
                                log_fname = "en_ta_prompt_fewshot",
                                result_fname = "")
```

```
name = ES
log_dir = Prompts/ES/fewshot_prompting
res_dir =  Results/ES/fewshot_prompting
attempting to open /content/eng_Latn.devtest
attempting to open /content/tam_Taml.devtest

100%|      | 1012/1012 [00:05<00:00, 179.61it/s]
```

```
[ ]: EXPERIMENT_EX_bilingual_prompting(input_file = "/content/eng_Latn.devtest",
                                label_files = ["/content/tam_Taml.devtest"],
                                fewshot_input_file = "/content/eng_Latn.dev",
                                fewshot_label_files = ["/content/tam_Taml.dev"],
                                translation_map = load_map("/content/
   ↪en_ta_mapping.pickle"),
                                language = "Tamil",
                                log_dir = "Prompts",
                                result_dir = " Results",
                                log_fname = "en_ta_prompt_bilingual",
                                result_fname = "")
```

```
name = ES
log_dir = Prompts/ES/fewshot_prompting
res_dir =  Results/ES/fewshot_prompting
attempting to open /content/eng_Latn.devtest
attempting to open /content/tam_Taml.devtest

100%|      | 1012/1012 [00:00<00:00, 4620.19it/s]
```

4.4 English –> Telugu

```
[ ]: EXPERIMENT_EX_fewshot_prompting(input_file = "/content/eng_Latn.devtest",
                                label_files = ["/content/tel_Telu.devtest"],
                                fewshot_input_file = "/content/eng_Latn.dev",
                                fewshot_label_files = ["/content/tel_Telu.dev"],
                                translation_map = load_map("/content/
   ↪en_te_mapping.pickle"),
                                language = "Telugu",
                                log_dir = "Prompts",
                                result_dir = " Results",
                                log_fname = "en_te_prompt_fewshot",
                                result_fname = "")
```

```
name = ES
log_dir = Prompts/ES/fewshot_prompting
res_dir =  Results/ES/fewshot_prompting
```

```
attempting to open /content/eng_Latn.devtest
attempting to open /content/tel_Telu.devtest

100%|     | 1012/1012 [00:04<00:00, 232.10it/s]
```

```
[ ]: EXPERIMENT_EX_bilingual_prompting(input_file = "/content/eng_Latn.devtest",
                                       label_files = ["/content/tel_Telu.devtest"],
                                       fewshot_input_file = "/content/eng_Latn.dev",
                                       fewshot_label_files = ["/content/tel_Telu.dev"],
                                       translation_map = load_map("/content/
      ↪en_te_mapping.pickle"),
                                       language = "Telugu",
                                       log_dir = "Prompts",
                                       result_dir = " Results",
                                       log_fname = "en_te_prompt_bilingual",
                                       result_fname = "")
```

```
name = ES
log_dir = Prompts/ES/fewshot_prompting
res_dir =  Results/ES/fewshot_prompting
attempting to open /content/eng_Latn.devtest
attempting to open /content/tel_Telu.devtest

100%|     | 1012/1012 [00:00<00:00, 2434.23it/s]
```