mainexperiment-prerequisites

November 29, 2023

1 Untokenize Word-Order-Shuffler results

[ ]:

```
original_f = 'flores_dev_english_svo'
tokenized_f = 'flores_dev_english_vso'

def untokenize_file(original_f, tokenized_f):
  with open(f'prev_{tokenized_f}', 'w') as f, open(tokenized_f, 'r') as g:
    f.write(g.read())

  untokenized_text = ""
  with open(original_f, 'r') as f, open(tokenized_f, 'r') as g:
    zipped = zip(f.readlines(), g.readlines())
    for original, tokenized_text in zipped:
      untokenized_text += untokenize(original.strip(), tokenized_text.strip())␣
 ↪+ ' \n'

  with open(tokenized_f, 'w') as f:
    f.write(untokenized_text)

untokenize_file(original_f, tokenized_f)
```

[ ]:

```
original = "During the 1976 selections advised then Carter he on foreign policy␣
 ↪, served as National Security Advisor ( NSA ) from 1977 to 1981 , succeeding␣
 ↪Henry Kissinger ."

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 recombine(list_of_str):
  import string
  ret = ""
  prev = ""
  for token in list_of_str:
    if prev == "-":
      ret = ret[:-1] + token + ' '
    elif token in string.punctuation:
      ret = ret[:-1] + token + ' '
    else:
      ret += token + ' '
    prev = token
  return ret[:-1]

def recombine_test(list_of_str):
  def has_punctuation(input_string):
      # Define a set of punctuation characters
      punctuation = set('!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~')
      # Iterate through the characters in the string
      for char in input_string:
          if char in punctuation:
              return True
      return False


  import string
  ret = ""
  prev = ""
  for token in list_of_str:
    if prev == "-":
      ret = ret[:-1] + token + ' '
    elif has_punctuation(token):
    # elif token in string.punctuation:
      ret = ret[:-1] + token + ' '
    else:
      ret += token + ' '
    prev = token
  return ret[:-1]
```

```
def untokenize(text):
  return recombine_test(tokenize(text))

print(f"tokenized = {tokenize(original)}")
print(original)
print(untokenize(original))
```

```
tokenized = ['During', 'the', '1976', 'selections', 'advised', 'then', 'Carter',
'he', 'on', 'foreign', 'policy', ',', 'served', 'as', 'National', 'Security',
'Advisor', '(', 'NSA', ')', 'from', '1977', 'to', '1981', ',', 'succeeding',
'Henry', 'Kissinger', '.']
During the 1976 selections advised then Carter he on foreign policy , served as
National Security Advisor ( NSA ) from 1977 to 1981 , succeeding Henry Kissinger
.
During the 1976 selections advised then Carter he on foreign policy, served as
National Security Advisor( NSA) from 1977 to 1981, succeeding Henry Kissinger.
```

2 Word-level noise maker

Noising: 1. A. 5 percent of character level bigram 2. B. Compounding

Cognates: 1. C1. German 2. C2. Portuguese 3. D1. Afrikaans 4. D2. Galician

List of Experiments: 1. A 2. B 3. B + A 4. C/Dx + A 5. C/Dx + B 6. C/Dx + B + A

2.1 A. Noising

Using 5% of character level bigram (bigram_5p.xlsx)

[ ]:
```
import pandas as pd
df = pd.read_excel('bigram_5p.xlsx')
translation_dict = df.set_index('Original')['Translation'].to_dict()

corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
 ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
 ↪'flores_english_svo'
]

A_corpora = {}

def obtain_mapping(refs_file, translation_dict, default_noising_map = {}):
  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

  noising_map = default_noising_map
  for ref_file in refs_file:
    with open(ref_file, 'r') as f:
      for sentence in f.readlines():
        tokens = tokenize(sentence)
        split_2 = [[token[i:i+2] for i in range(0, len(token), 2)] for token in tokens]
        for split_idx in range(len(split_2)):
          split = split_2[split_idx]
          for i in range(len(split)):
            if split[i] in translation_dict:
              split[i] = translation_dict[split[i]]

        corrupted_tokens = [''.join(subtokens) for subtokens in split_2]
        for ori, trans in zip(tokens, corrupted_tokens):
          if noising_map.get(ori, 0) == 0:
            noising_map[ori] = trans
          elif noising_map[ori] != trans:
            print(f"noising_map[{ori}] = {noising_map[ori]}, not {trans}")
  return noising_map

def A_noising(corpus, noising_map):
  res = ''
  for sentence in corpus:
    tokens = tokenize(sentence)
    for i in range(len(tokens)):
      tokens[i] = noising_map[tokens[i]]

    corrupted_sentence = recombine(tokens)
    res += corrupted_sentence +'\n'
  return res

# refs_file are one from devtest and one from dev, it doesn't matter which word-order is used
```

```
refs_file = ["flores_english_svo", "flores_dev_english_svo"]
noising_map = obtain_mapping(corpora, translation_dict)

# for corpus in corpora:
#   with open(corpus, 'r') as f:
#     A_corpora[f'A_{corpus}'] = A_noising(f.readlines(), noising_map)

# for filename in A_corpora.keys():
#   with open(filename, 'w') as f:
#     f.write(A_corpora[filename])

# import pickle
# with open('A_noising_map.pickle', 'wb') as f:
#   pickle.dump(noising_map, f)
```

2.2 B. Compounding

Only uses the top 5 percent bigrams found from wikipedia 100K dataset

[ ]:

```
# Compounder Code
import random

class CompoundNoise:
        def __init__(self):
                self.mapping = {}
                self.reverse_mapping = {}
                self.banned = []

        def set_map(self, mapping):
                self.mapping = mapping

        def set_reverse_map(self, reverse_mapping):
                self.reverse_mapping = reverse_mapping

        def set_banned(self, banned):
                self.banned = banned

        def clear_map(self):
                self.mapping = {}
                self.reverse_mapping = {}

        def get_map(self):
                return self.mapping

        def get_reverse_map(self):
                return self.reverse_mapping
        def get_banned(self):
                return self.banned

        def compound_token(self, s1, s2):
                if (s1, s2) in self.mapping:
                        return self.mapping[(s1, s2)]

                # try blending
                blended = self.generate_blend(s1, s2)
                if blended != -1:
                        self.mapping[(s1, s2)] = blended
                        self.reverse_mapping[blended] = (s1, s2)
                        return blended

                portmanteaued = self.generate_portmanteau(s1, s2)
                if portmanteaued != -1:
                        self.mapping[(s1, s2)] = portmanteaued
                        self.reverse_mapping[portmanteaued] = (s1, s2)
                        return portmanteaued

                # perform no compounding
                return s1 + ' ' + s2

        # Helper function
        def get_indices(self, lst, target_element):
                return [index for index, element in enumerate(lst) if element␣
↪== target_element]

        def find_common_character(self, word1, word2):
                chars = []
                for char in word1:
                        if char in word2:
                                chars.append(char)
                return chars

        def generate_blend(self, word1, word2):
                common_chars = list(set(self.find_common_character(word1,␣
↪word2)))
                candidates = []
                len_diff = []
                for common_char in common_chars:
                        if common_char:
                                index1 = self.get_indices(word1, common_char)
                                index2 = self.get_indices(word2, common_char)

                                average_length = (len(word1) + len(word2))//2
                                for idx1 in index1:
```

```
                    for idx2 in index2:
                        new_word = word1[:idx1] +␣
↪word2[idx2:]
                        if new_word in self.mapping or␣
↪new_word in self.banned:
                            continue
                        if abs(len(new_word) -␣
↪average_length) <= 3:
                            candidates.
↪append(new_word)
                            len_diff.
↪append(abs(len(new_word) - average_length))

        mini = 999
        mini_idx = None
        for i in range(len(candidates)):
            if len_diff[i] < mini:
                mini = len_diff[i]
                mini_idx = i

        if mini_idx is not None:
            return candidates[mini_idx]
        return -1

    def generate_portmanteau(self, word1, word2):
        portmanteau_type = [1] * 2 + [2] * 1 + [3] * 2 + [4] * 5
        random.shuffle(portmanteau_type)

        choice = random.choice(portmanteau_type)
        # Type 1: Full Append (20% chance)
        # Example: [basket] + [ball] = basketball
        res = ""
        if choice == 1:
            res = word1 + word2

        # Type 2: word1 + half end of word2 (10% chance)
        # Example: [guess] + es[timate] = guesstimate
        elif choice == 2:
            res = word1 + word2[int(len(word2)/2):]

        # Type 3: half first of word1 + half first of word2 (20% chance)
        # Example: [sit]uation + [com]edy = sitcom
        elif choice == 3:
            res = word1[:int(len(word1)/2)] + word2[:int(len(word2)/
↪2)]
```

```
                # Type 4: half first of word1 + half second of word2 (50%␣
 ↪chance)
                # Example: [glam]orous + cam[ping] = glamping
                elif choice == 4:
                        res = word1[:int(len(word1)/2)] + word2[int(len(word2)/
 ↪2):]

                if res in self.mapping or res in self.banned:
                        return -1
                return res
```

[ ]:

```
Compounder = CompoundNoise()

# Ban words that are already in the A_noising_map's keys
import pickle
mapping = {}
with open(f'A_noising_map.pickle', 'rb') as f:
  mapping = pickle.load(f)

original = []
for ori, trans in mapping.items():
  original.append(ori)

# Reminder, there are entries in original that translate into the same word
BANNED = list(set(original))

Compounder.set_banned(BANNED)
```

[ ]:

```
import pandas as pd
df = pd.read_excel('word_bigram_lowercase_5plus.xlsx')
df = df['bigram']

bigram_list = df.tolist()
bigram_list = [[word.split(',')[0][2:-1], word.split(',')[1][2:-2]] for word in␣
 ↪bigram_list]

for bigram in bigram_list:
  Compounder.compound_token(bigram[0], bigram[1])
```

[ ]:

```
# before B_compounding
len(Compounder.get_map())
```

[ ]:

```
9861
```

[ ]:

```
compound_map = Compounder.get_map()

import pickle
```

```
with open(f'B_compound_map.pickle', 'wb') as f:
  pickle.dump(compound_map, f)

with open(f'B_reverse_compound_map.pickle', 'wb') as f:
  pickle.dump(Compounder.get_reverse_map(), f)
```

[ ]:

```
corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
  ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
  ↪'flores_english_svo'
]

B_corpora = {}

def B_compounding(corpus, compound_map):
  res = ''
  for sentence in corpus:
    tokens = sentence.split(' ')
    if len(tokens) == 1:
      return str(tokens[0])

    bigrams = [(tokens[i], tokens[i+1]) for i in range(len(tokens) - 1)]

    for bigram_idx in range(len(bigrams)):
      if bigrams[bigram_idx] in compound_map:
        bigrams[bigram_idx] = (compound_map[bigrams[bigram_idx]], '')

        if bigram_idx == 0:
          bigrams[1] = ('', bigrams[1][1])
        elif bigram_idx == len(bigrams) - 1:
          bigrams[len(bigrams) - 2] = (bigrams[len(bigrams) - 2][0], '')
        else:
          bigrams[bigram_idx-1] = (bigrams[bigram_idx-1][0], '')
          bigrams[bigram_idx+1] = ('', bigrams[bigram_idx+1][1])

    reconstruct = []
    for i in range(len(bigrams)):
      if i != len(bigrams) - 1:
        if bigrams[i][0] != '':
          reconstruct.append(bigrams[i][0])
      else:
        if bigrams[i][0] != '':
          reconstruct.append(bigrams[i][0])
          reconstruct.append(bigrams[i][1])
        else:
          reconstruct.append(bigrams[i][1])
    reconstruct = ' '.join(reconstruct)
    res += reconstruct

  return res

import pickle
compound_map = {}
with open('B_compound_map.pickle', 'rb') as f:
  compound_map = pickle.load(f)

for corpus in corpora:
  with open(corpus, 'r') as f:
    B_corpora[f'B_{corpus}'] = B_compounding(f.readlines(), compound_map)

for filename in B_corpora.keys():
  with open(f'temp_{filename}', 'w') as f:
    f.write(B_corpora[filename])
```

2.3 A + B

1. Get B files
2. Noise using A_mapping_original.pickle

```
[ ]: # @title Default title text
import pickle
corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
  ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
  ↪'flores_english_svo'
]

A_noising_map = {}
with open('A_noising_map.pickle', 'rb') as f:
  A_noising_map = pickle.load(f)

refs_file = [f"B_{corpus}" for corpus in corpora]
AB_noising_map = obtain_mapping(refs_file, translation_dict, A_noising_map)

AB_corpora = {}
for corpus in corpora:
  with open(f'B_{corpus}', 'r') as f:
    AB_corpora[f'AB_{corpus}'] = A_noising(f.readlines(), AB_noising_map)

with open('AB_noising_map.pickle', 'wb') as f:
  pickle.dump(AB_noising_map, f)
```

```
for filename in AB_corpora.keys():
  with open(filename, 'w') as f:
    f.write(AB_corpora[filename])
```

[ ]:

```

```

2.4 C/Dx

Get mapping first, use google translate (Created on 22/09/2023) for the four following language:

```
C1. German (de)
C2. Portuguese (pt)
D1. Afrikaans (af)
D2. Galician (gl)
```

[ ]:

```
!gcloud auth application-default login
!gcloud auth application-default set-quota-project resolute-parity-392608
!gcloud auth login
```

[ ]:

```
import requests
import subprocess
import json

def romanize_text(src, contents):
    # Define the data you want to send in the POST request
    data = {
        'source_language_code': src,
        'contents': contents
    }

    # Define the file path where you want to save the JSON data
    file_path = 'request.json'

    # Write the dictionary to the JSON file
    with open(file_path, 'w') as json_file:
        json.dump(data, json_file, indent=4)

    command = """
    curl -X POST \
        -H "Authorization: Bearer $(gcloud auth print-access-token)" \
        -H "x-goog-user-project: resolute-parity-392608" \
        -H "Content-Type: application/json; charset=utf-8" \
        -d @request.json \
        "https://translation.googleapis.com/v3/projects/resolute-parity-392608/
 ↪locations/us-central1:romanizeText" \
        -o "romanized.json"
    """
    subprocess.run(command, shell=True, text=True, stdout=subprocess.PIPE)

    romanized_text_list = []
    with open('romanized.json', 'r') as f:
      romanized_texts = dict(json.load(f))['romanizations']

      for roman_text in romanized_texts:
        romanized_text_list.append(roman_text['romanizedText'])

    return romanized_text_list

# romanize_text('ja', ['   ', '         '])

def translate_text(target: str, source: str, text: str) -> dict:
    """Translates text into the target language.

    Target must be an ISO 639-1 language code.
    See https://g.co/cloud/translate/v2/translate-reference#supported_languages
    """
    from google.cloud import translate_v2 as translate

    translate_client = translate.Client()

    if isinstance(text, bytes):
        text = text.decode("utf-8")

    result = translate_client.translate(text, target_language=target,␣
 ↪source_language=source)

    non_roman = {
      'ar': 'Arabic',
      'am': 'Amharic',
      'bn': 'Bengali',
      'be': 'Belarusian',
      'hi': 'Hindi',
      'ja': 'Japanese',
      'my': 'Myanmar',
      'ru': 'Russian',
      'sr': 'Serbian',
      'uk': 'Ukrainian'
    }

    if target in non_roman.keys():
      result = romanize_text(target, result['translatedText'])
      return result
    return result['translatedText']
```

[ ]:

```
import pickle
en_words = []
en_dicts = {}

with open('A_noising_map.pickle', 'rb') as f:
  en_dicts = pickle.load(f)

en_words = list(en_dicts.keys())
```

[ ]:

```
from tqdm import tqdm

import random

prob = 0.2 # 20% of the dictionary will be cognated
flag_translate = [1 if random.random() < prob else 0 for i in␣
 ↪range(len(en_words))]

i = 0
translate_map = {}
for word in en_words:
  translate_map[word] = flag_translate[i]
  i += 1

C1_mapping = {}
C2_mapping = {}
D1_mapping = {}
D2_mapping = {}

languages = ['de', 'pt', 'af', 'gl']

def transform_map(translate_map, tgt):
  result_map = {}
  for word, flag in tqdm(translate_map.items()):
    if flag:
      result_map[word] = translate_text(tgt, 'en', word)
    else:
      result_map[word] = word
  return result_map
```

[ ]:

```
import pickle
with open('translate_map.pickle', 'wb') as f:
  pickle.dump(translate_map, f)
```

[ ]:

```
import pickle
from google.colab import files

C1_mapping = transform_map(translate_map, 'de')
```

```
with open('C1_mapping.pickle', 'wb') as file:
    pickle.dump(C1_mapping, file)
files.download('C1_mapping.pickle')
```

```
100%|      | 10608/10608 [42:41<00:00,  4.14it/s]

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

```
[ ]: import pickle
     from google.colab import files

     C2_mapping = transform_map(translate_map, 'pt')
     with open('C2_mapping.pickle', 'wb') as file:
         pickle.dump(C2_mapping, file)
     files.download('C2_mapping.pickle')
```

```
100%|      | 10608/10608 [41:58<00:00,  4.21it/s]

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

```
[ ]: import pickle
     from google.colab import files

     D1_mapping = transform_map(translate_map, 'af')
     with open('D1_mapping.pickle', 'wb') as file:
         pickle.dump(D1_mapping, file)
     files.download('D1_mapping.pickle')
```

```
100%|      | 10608/10608 [41:33<00:00,  4.25it/s]

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

```
[ ]: import pickle
     from google.colab import files

     D2_mapping = transform_map(translate_map, 'gl')
     with open('D2_mapping.pickle', 'wb') as file:
         pickle.dump(D2_mapping, file)
     files.download('D2_mapping.pickle')
```

```
100%|      | 10608/10608 [42:46<00:00,  4.13it/s]

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

2.4.1 Mapping and Translate

[ ]:

```
corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
  ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
  ↪'flores_english_svo'
]

import pickle
with open(f'C1_mapping.pickle', 'rb') as f:
  C1_map = pickle.load(f)

with open(f'C2_mapping.pickle', 'rb') as f:
  C2_map = pickle.load(f)

with open(f'D1_mapping.pickle', 'rb') as f:
  D1_map = pickle.load(f)

with open(f'D2_mapping.pickle', 'rb') as f:
  D2_map = pickle.load(f)
```

[ ]:

```
def translate_corpora(code, files, mapping):
  for file_ in files:
    res = ''
    with open(file_, 'r') as f:
      for sentence in f:
        tokens = tokenize(sentence)
        for i in range(len(tokens)):
          tokens[i] = mapping.get(tokens[i], tokens[i])

        corrupted_sentence = recombine(tokens)
        res += corrupted_sentence +'\n'

    with open(f"{code}_{file_}", 'w') as f:
      f.write(res)

translate_corpora('C1', corpora, C1_map)
translate_corpora('C2', corpora, C2_map)
translate_corpora('D1', corpora, D1_map)
translate_corpora('D2', corpora, D2_map)
```

2.5 (C/D + A). Cognates with noising

2.5.1 A. Function:

1. A_noising
2. obtain_mapping

```
[ ]: def obtain_mapping(refs_file, translation_dict, default_noising_map = {}, allow_update = False):
       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

       noising_map = default_noising_map
       for ref_file in refs_file:
         with open(ref_file, 'r') as f:
           for sentence in f.readlines():
             tokens = tokenize(sentence)
             split_2 = [[token[i:i+2] for i in range(0, len(token), 2)] for token in tokens]
             for split_idx in range(len(split_2)):
               split = split_2[split_idx]
               for i in range(len(split)):
                 if split[i] in translation_dict:
                   split[i] = translation_dict[split[i]]

             corrupted_tokens = [''.join(subtokens) for subtokens in split_2]
             for ori, trans in zip(tokens, corrupted_tokens):
               if noising_map.get(ori, 0) == 0 or noising_map.get(ori, 0) == trans:
                 noising_map[ori] = trans
               elif noising_map[ori] != trans and allow_update:
                 noising_map[ori] = trans
               else:
                 print(f"noising_map[{ori}] = {noising_map[ori]}, not {trans} | However, update is not allowed")
       return noising_map

     def A_noising(corpus, noising_map):
       res = ''
       for sentence in corpus:
```

```
    tokens = tokenize(sentence)
    for i in range(len(tokens)):
      tokens[i] = noising_map[tokens[i]]

    corrupted_sentence = recombine(tokens)
    res += corrupted_sentence +'\n'
  return res
```

[ ]:

```
def translate_corpora(code, files, mapping):
  for file_ in files:
    res = ''
    with open(file_, 'r') as f:
      for sentence in f:
        tokens = tokenize(sentence)
        for i in range(len(tokens)):
          tokens[i] = mapping[tokens[i]]

        corrupted_sentence = recombine(tokens)
        res += corrupted_sentence +'\n'

    with open(f"{code}_{file_}", 'w') as f:
      f.write(res)
```

2.5.2 C1 + A. German Cognated + Noising

[ ]:

```
import pickle

corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
  ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
  ↪'flores_english_svo'
]

translate_corpora('C1', corpora, C1_map)

C1_noise_map = {}
with open('C1_mapping.pickle', 'rb') as f:
  C1_noise_map = pickle.load(f) # Basically, C1_noise_map = C1_map

refs_file = [f"C1_{corpus}" for corpus in corpora]
C1A_noising_map = obtain_mapping(refs_file, translation_dict, C1_noise_map,␣
  ↪allow_update = True)

C1A_corpora = {}
for corpus in corpora:
  with open(f'C1_{corpus}', 'r') as f:
    C1A_corpora[f'C1A_{corpus}'] = A_noising(f.readlines(), C1A_noising_map)

with open('C1A_noising_map.pickle', 'wb') as f:
  pickle.dump(C1A_noising_map, f)

for filename in C1A_corpora.keys():
  with open(filename, 'w') as f:
    f.write(C1A_corpora[filename])
```

[ ]:

```
C1A_noising_map
```

2.5.3 C2 + A. Portuguese Cognated + Noising

[ ]:

```
import pickle

corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
  ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
  ↪'flores_english_svo'
]

translate_corpora('C2', corpora, C2_map)

C2_noise_map = {}
with open('C2_mapping.pickle', 'rb') as f:
  C2_noise_map = pickle.load(f)

refs_file = [f"C2_{corpus}" for corpus in corpora]
C2A_noising_map = obtain_mapping(refs_file, translation_dict, C2_noise_map,␣
  ↪allow_update = True)

C2A_corpora = {}
for corpus in corpora:
  with open(f'C2_{corpus}', 'r') as f:
    C2A_corpora[f'C2A_{corpus}'] = A_noising(f.readlines(), C2A_noising_map)

with open('C2A_noising_map.pickle', 'wb') as f:
  pickle.dump(C2A_noising_map, f)

for filename in C2A_corpora.keys():
  with open(filename, 'w') as f:
    f.write(C2A_corpora[filename])
```

2.5.4 D1 + A. Afrikaans Cognated + Noising

[ ]:

```
import pickle

corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
  ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
  ↪'flores_english_svo'
]

translate_corpora('D1', corpora, D1_map)

D1_noise_map = {}
with open('D1_mapping.pickle', 'rb') as f:
  D1_noise_map = pickle.load(f)

refs_file = [f"D1_{corpus}" for corpus in corpora]
D1A_noising_map = obtain_mapping(refs_file, translation_dict, D1_noise_map,␣
  ↪allow_update = True)

D1A_corpora = {}
for corpus in corpora:
  with open(f'D1_{corpus}', 'r') as f:
    D1A_corpora[f'D1A_{corpus}'] = A_noising(f.readlines(), D1A_noising_map)

with open('D1A_noising_map.pickle', 'wb') as f:
  pickle.dump(D1A_noising_map, f)

for filename in D1A_corpora.keys():
  with open(filename, 'w') as f:
    f.write(D1A_corpora[filename])
```

2.5.5 D2 + A. Galician Cognated + Noising

[ ]:

```
import pickle

corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
  ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
  ↪'flores_english_svo'
]

translate_corpora('D2', corpora, D1_map)

D2_noise_map = {}
```

```
with open('D2_mapping.pickle', 'rb') as f:
  D2_noise_map = pickle.load(f)

refs_file = [f"D2_{corpus}" for corpus in corpora]
D2A_noising_map = obtain_mapping(refs_file, translation_dict, D2_noise_map,␣
 ↪allow_update = True)

D2A_corpora = {}
for corpus in corpora:
  with open(f'D2_{corpus}', 'r') as f:
    D2A_corpora[f'D2A_{corpus}'] = A_noising(f.readlines(), D2A_noising_map)

with open('D2A_noising_map.pickle', 'wb') as f:
  pickle.dump(D2A_noising_map, f)

for filename in D2A_corpora.keys():
  with open(filename, 'w') as f:
    f.write(D2A_corpora[filename])
```

2.6 (C/D + B). Cognates with Compounding

2.6.1 B. Function:

[ ]:

```
# Compounder Code
import random

class CompoundNoise:
        def __init__(self):
                self.mapping = {}
                self.reverse_mapping = {}
                self.banned = []

        def set_map(self, mapping):
                self.mapping = mapping

        def set_reverse_map(self, reverse_mapping):
                self.reverse_mapping = reverse_mapping

        def set_banned(self, banned):
                self.banned = banned

        def clear_map(self):
                self.mapping = {}
                self.reverse_mapping = {}

        def get_map(self):
                return self.mapping
        def get_reverse_map(self):
                return self.reverse_mapping

        def get_banned(self):
                return self.banned

        def compound_token(self, s1, s2):
                if (s1, s2) in self.mapping:
                        return self.mapping[(s1, s2)]

                # try blending
                blended = self.generate_blend(s1, s2)
                if blended != -1:
                        self.mapping[(s1, s2)] = blended
                        self.reverse_mapping[blended] = (s1, s2)
                        return blended

                portmanteaued = self.generate_portmanteau(s1, s2)
                if portmanteaued != -1:
                        self.mapping[(s1, s2)] = portmanteaued
                        self.reverse_mapping[portmanteaued] = (s1, s2)
                        return portmanteaued

                # perform no compounding
                return s1 + ' ' + s2

        # Helper function
        def get_indices(self, lst, target_element):
                return [index for index, element in enumerate(lst) if element␣
↪== target_element]

        def find_common_character(self, word1, word2):
                chars = []
                for char in word1:
                        if char in word2:
                                chars.append(char)
                return chars

        def generate_blend(self, word1, word2):
                common_chars = list(set(self.find_common_character(word1,␣
↪word2)))
                candidates = []
                len_diff = []
                for common_char in common_chars:
                        if common_char:
                                index1 = self.get_indices(word1, common_char)
                index2 = self.get_indices(word2, common_char)

                average_length = (len(word1) + len(word2))//2
                for idx1 in index1:
                    for idx2 in index2:
                        new_word = word1[:idx1] +␣
↪word2[idx2:]
                        if new_word in self.mapping or␣
↪new_word in self.banned:
                            continue
                        if abs(len(new_word) -␣
↪average_length) <= 3:
                            candidates.
↪append(new_word)
                            len_diff.
↪append(abs(len(new_word) - average_length))

        mini = 999
        mini_idx = None
        for i in range(len(candidates)):
            if len_diff[i] < mini:
                mini = len_diff[i]
                mini_idx = i

        if mini_idx is not None:
            return candidates[mini_idx]
        return -1

    def generate_portmanteau(self, word1, word2):
        portmanteau_type = [1] * 2 + [2] * 1 + [3] * 2 + [4] * 5
        random.shuffle(portmanteau_type)

        choice = random.choice(portmanteau_type)
        # Type 1: Full Append (20% chance)
        # Example: [basket] + [ball] = basketball
        res = ""
        if choice == 1:
            res = word1 + word2

        # Type 2: word1 + half end of word2 (10% chance)
        # Example: [guess] + es[timate] = guesstimate
        elif choice == 2:
            res = word1 + word2[int(len(word2)/2):]

        # Type 3: half first of word1 + half first of word2 (20% chance)
        # Example: [sit]uation + [com]edy = sitcom
        elif choice == 3:
```

```
                        res = word1[:int(len(word1)/2)] + word2[:int(len(word2)/
↪2)]

                # Type 4: half first of word1 + half second of word2 (50%␣
↪chance)
                # Example: [glam]orous + cam[ping] = glamping
                elif choice == 4:
                        res = word1[:int(len(word1)/2)] + word2[int(len(word2)/
↪2):]

                if res in self.mapping or res in self.banned:
                        return -1
                return res
```

[ ]:

```
def B_compounding(corpus, compound_map):
  res = ''
  for sentence in corpus:
    tokens = sentence.split(' ')
    if len(tokens) == 1:
      return str(tokens[0])

    bigrams = [(tokens[i], tokens[i+1]) for i in range(len(tokens) - 1)]

    for bigram_idx in range(len(bigrams)):
      if bigrams[bigram_idx] in compound_map:
        bigrams[bigram_idx] = (compound_map[bigrams[bigram_idx]], '')

        if bigram_idx == 0:
          bigrams[1] = ('', bigrams[1][1])
        elif bigram_idx == len(bigrams) - 1:
          bigrams[len(bigrams) - 2] = (bigrams[len(bigrams) - 2][0], '')
        else:
          bigrams[bigram_idx-1] = (bigrams[bigram_idx-1][0], '')
          bigrams[bigram_idx+1] = ('', bigrams[bigram_idx+1][1])

    reconstruct = []
    for i in range(len(bigrams)):
      if i != len(bigrams) - 1:
        if bigrams[i][0] != '':
          reconstruct.append(bigrams[i][0])
      else:
        if bigrams[i][0] != '':
          reconstruct.append(bigrams[i][0])
          reconstruct.append(bigrams[i][1])
        else:
          reconstruct.append(bigrams[i][1])
    reconstruct = ' '.join(reconstruct)
    res += reconstruct

  return res
```

2.6.2 C1 + B German Cognates + Compounding

[ ]:

```
Compounder = CompoundNoise()

# Ban words that are already in the A_noising_map's keys
import pickle
mapping = {}
with open(f'C1_mapping.pickle', 'rb') as f:
  mapping = pickle.load(f)

# Because of Cognates, we do not want the Compounding resulting in an already␣
  ↪existing
# word in either english or the other language
original = []
for ori, trans in mapping.items():
  original.append(ori)
  original.append(trans)

# Reminder, there are entries in original that translate into the same word
BANNED = list(set(original))

Compounder.set_banned(BANNED)
```

[ ]:

```
import pandas as pd
df = pd.read_excel('word_bigram_lowercase_5plus.xlsx')
df = df['bigram']

bigram_list = df.tolist()
bigram_list = [[word.split(',')[0][2:-1], word.split(',')[1][2:-2]] for word in␣
  ↪bigram_list]

for bigram in bigram_list:
  for i in range(len(bigram)):
    if bigram[i] in C1_map:
      bigram[i] = C1_map[bigram[i]]

for bigram in bigram_list:
  Compounder.compound_token(bigram[0], bigram[1])

compound_map_C1 = Compounder.get_map()

for corpus in corpora:
  with open(f'C1_{corpus}', 'r') as f, open(f'C1B_{corpus}', 'w') as g:
    g.write(B_compounding(f.readlines(), compound_map_C1))

import pickle
with open(f'C1B_compound_map.pickle', 'wb') as f:
  pickle.dump(compound_map_C1, f)

with open(f'C1B_combined_map.pickle', 'wb') as f:
  pickle.dump({**compound_map_C1, **C1_map} ,f)
```

2.6.3 C2 + B Portuguese Cognates + Compounding

[ ]:

```
import pandas as pd
df = pd.read_excel('word_bigram_lowercase_5plus.xlsx')
df = df['bigram']

bigram_list = df.tolist()
bigram_list = [[word.split(',')[0][2:-1], word.split(',')[1][2:-2]] for word in␣
 ↪bigram_list]

for bigram in bigram_list:
  for i in range(len(bigram)):
    if bigram[i] in C2_map:
      bigram[i] = C2_map[bigram[i]]

for bigram in bigram_list:
  Compounder.compound_token(bigram[0], bigram[1])

compound_map_C2 = Compounder.get_map()

for corpus in corpora:
  with open(f'C2_{corpus}', 'r') as f, open(f'C2B_{corpus}', 'w') as g:
    g.write(B_compounding(f.readlines(), compound_map_C2))

import pickle
with open(f'C2B_compound_map.pickle', 'wb') as f:
  pickle.dump(compound_map_C2, f)

with open(f'C2B_combined_map.pickle', 'wb') as f:
  pickle.dump({**compound_map_C2, **C2_map} ,f)
```

2.6.4 D1 + B Afrikaans Cognates + Compounding

[ ]:

```
import pandas as pd
df = pd.read_excel('word_bigram_lowercase_5plus.xlsx')
df = df['bigram']
```

```
bigram_list = df.tolist()
bigram_list = [[word.split(',')[0][2:-1], word.split(',')[1][2:-2]] for word in␣
 ↪bigram_list]

for bigram in bigram_list:
  for i in range(len(bigram)):
    if bigram[i] in D1_map:
      bigram[i] = D1_map[bigram[i]]

for bigram in bigram_list:
  Compounder.compound_token(bigram[0], bigram[1])

compound_map_D1 = Compounder.get_map()

for corpus in corpora:
  with open(f'D1_{corpus}', 'r') as f, open(f'D1B_{corpus}', 'w') as g:
    g.write(B_compounding(f.readlines(), compound_map_D1))

import pickle
with open(f'D1B_compound_map.pickle', 'wb') as f:
  pickle.dump(compound_map_D1, f)

with open(f'D1B_combined_map.pickle', 'wb') as f:
  pickle.dump({**compound_map_D1, **D1_map} ,f)
```

2.6.5 D2 + B Galician Cognates + Compounding

[ ]:

```
import pandas as pd
df = pd.read_excel('word_bigram_lowercase_5plus.xlsx')
df = df['bigram']

bigram_list = df.tolist()
bigram_list = [[word.split(',')[0][2:-1], word.split(',')[1][2:-2]] for word in␣
 ↪bigram_list]

for bigram in bigram_list:
  for i in range(len(bigram)):
    if bigram[i] in D2_map:
      bigram[i] = D2_map[bigram[i]]

for bigram in bigram_list:
  Compounder.compound_token(bigram[0], bigram[1])

compound_map_D2 = Compounder.get_map()
```

```
for corpus in corpora:
  with open(f'D2_{corpus}', 'r') as f, open(f'D2B_{corpus}', 'w') as g:
    g.write(B_compounding(f.readlines(), compound_map_D2))

import pickle
with open(f'D2B_compound_map.pickle', 'wb') as f:
  pickle.dump(compound_map_D2, f)

with open(f'D2B_combined_map.pickle', 'wb') as f:
  pickle.dump({**compound_map_D2, **D2_map} ,f)
```

2.7 (C/D + B + A). Cognates with Compounding THEN Noising

From the results of C/D + B, get **C/Dx_noising_map**. Use it as default mapping for obtain_mapping function. Allow updates = True, save the mapping in **C/Dx_BA_noising_map.pickle**. Noise the C/D + B corpus.

This means: We utlized **C/Dx_map** to obtain **C/DxA_noising_map.pickle**, which we then use to obtain **C/DxBA_noising_map.pickle**.

We also need **C/Dx_compound_map.pickle** (NOT COMBINED_MAP) to then reconstruct English corpus into (C/D + B) corpus first. To make things easier, we can create **C/DxBA_combined_map.pickle**, consisting of **C/Dx_compound_map.pickle + C/DxBA_noising_map.pickle.**

2.7.1 BA. Function:

–> Actually, we only need the A functions since this is done after (C/D + B)

[ ]:

```
def obtain_mapping(refs_file, translation_dict, default_noising_map = {},␣
 ↪allow_update = False):
  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
  noising_map = default_noising_map
  for ref_file in refs_file:
    with open(ref_file, 'r') as f:
      for sentence in f.readlines():
        tokens = tokenize(sentence)
        split_2 = [[token[i:i+2] for i in range(0, len(token), 2)] for token in␣
 ↪tokens]
        for split_idx in range(len(split_2)):
          split = split_2[split_idx]
          for i in range(len(split)):
            if split[i] in translation_dict:
              split[i] = translation_dict[split[i]]

        corrupted_tokens = [''.join(subtokens) for subtokens in split_2]
        for ori, trans in zip(tokens, corrupted_tokens):
          if noising_map.get(ori, 0) == 0 or noising_map.get(ori, 0) == trans:
            noising_map[ori] = trans
          elif noising_map[ori] != trans and allow_update:
            noising_map[ori] = trans
          else:
            print(f"noising_map[{ori}] = {noising_map[ori]}, not {trans} |␣
 ↪However, update is not allowed")
  return noising_map

def A_noising(corpus, noising_map):
  res = ''
  for sentence in corpus:
    tokens = tokenize(sentence)
    for i in range(len(tokens)):
      tokens[i] = noising_map[tokens[i]]

    corrupted_sentence = recombine(tokens)
    res += corrupted_sentence +'\n'
  return res
```

2.7.2 Setup

[ ]:
```
corpora = [
    'flores_dev_english_sov', 'flores_dev_english_vos',␣
 ↪'flores_dev_english_vso', 'flores_dev_english_svo',
    'flores_english_sov', 'flores_english_vos', 'flores_english_vso',␣
 ↪'flores_english_svo'
]

import pickle
```

```
C1A_noising_map = {}
with open('C1A_noising_map.pickle', 'rb') as f:
  C1A_noising_map = pickle.load(f)

C2A_noising_map = {}
with open('C2A_noising_map.pickle', 'rb') as f:
  C2A_noising_map = pickle.load(f)

D1A_noising_map = {}
with open('D1A_noising_map.pickle', 'rb') as f:
  D1A_noising_map = pickle.load(f)

D2A_noising_map = {}
with open('D2A_noising_map.pickle', 'rb') as f:
  D2A_noising_map = pickle.load(f)

import pandas
df = pd.read_excel('bigram_5p.xlsx')
translation_dict = df.set_index('Original')['Translation'].to_dict()
```

[ ]:

```
import re

def replace_full_words(text, old_word, new_word):
    pattern = r'\b{}\b'.format(re.escape(old_word))
    result = re.sub(pattern, new_word, text)
    return result
```

2.7.3 C1 + B + A. German Cognates + Compounding + Noising

[ ]:

```
refs_file = [f"C1B_{corpus}" for corpus in corpora]
C1BA_noising_map = obtain_mapping(refs_file, translation_dict, C1A_noising_map,␣
 ↪allow_update = True)

for corpus in corpora:
  text = ''
  with open(f'C1B_{corpus}', 'r') as f:
    text = f.read()
    for clean, noise in C1BA_noising_map.items():
      if clean != noise:
        text = replace_full_words(text, clean, noise)

  with open(f'C1BA_{corpus}', 'w') as f:
    f.write(text)

import pickle
C1B_compound_map = {}
```

```
with open('C1B_compound_map.pickle', 'rb') as f:
  C1B_compound_map = pickle.load(f)

with open('C1BA_noising_map.pickle', 'wb') as f:
  pickle.dump(C1BA_noising_map, f)

with open('C1BA_combined_map.pickle', 'wb') as f:
  pickle.dump({**C1B_compound_map, **C1BA_noising_map}, f)
```

2.7.4 C2 + B + A. Portuguese Cognates + Compounding + Noising

[ ]:

```
refs_file = [f"C2B_{corpus}" for corpus in corpora]
C2BA_noising_map = obtain_mapping(refs_file, translation_dict, C2A_noising_map,␣
 ↪allow_update = True)

for corpus in corpora:
  text = ''
  with open(f'C2B_{corpus}', 'r') as f:
    text = f.read()
    for clean, noise in C2BA_noising_map.items():
      if clean != noise:
        text = replace_full_words(text, clean, noise)

  with open(f'C2BA_{corpus}', 'w') as f:
    f.write(text)

import pickle
C2B_compound_map = {}
with open('C2B_compound_map.pickle', 'rb') as f:
  C2B_compound_map = pickle.load(f)

with open('C2BA_noising_map.pickle', 'wb') as f:
  pickle.dump(C2BA_noising_map, f)

with open('C2BA_combined_map.pickle', 'wb') as f:
  pickle.dump({**C2B_compound_map, **C2BA_noising_map}, f)
```

2.7.5 D1 + B + A. Afrikaans Cognates + Compounding + Noising

[ ]:

```
refs_file = [f"D1B_{corpus}" for corpus in corpora]
D1BA_noising_map = obtain_mapping(refs_file, translation_dict, D1A_noising_map,␣
 ↪allow_update = True)

for corpus in corpora:
  text = ''
  with open(f'D1B_{corpus}', 'r') as f:
    text = f.read()
    for clean, noise in D1BA_noising_map.items():
      if clean != noise:
        text = replace_full_words(text, clean, noise)

  with open(f'D1BA_{corpus}', 'w') as f:
    f.write(text)

import pickle
D1B_compound_map = {}
with open('D1B_compound_map.pickle', 'rb') as f:
  D1B_compound_map = pickle.load(f)

with open('D1BA_noising_map.pickle', 'wb') as f:
  pickle.dump(D1BA_noising_map, f)

with open('D1BA_combined_map.pickle', 'wb') as f:
  pickle.dump({**D1B_compound_map, **D1BA_noising_map}, f)
```

2.7.6 D2 + B + A. Galician Cognates + Compounding + Noising

[ ]:

```
refs_file = [f"D2B_{corpus}" for corpus in corpora]
D2BA_noising_map = obtain_mapping(refs_file, translation_dict, D2A_noising_map,␣
 ↪allow_update = True)

for corpus in corpora:
  text = ''
  with open(f'D2B_{corpus}', 'r') as f:
    text = f.read()
    for clean, noise in D2BA_noising_map.items():
      if clean != noise:
        text = replace_full_words(text, clean, noise)

  with open(f'D2BA_{corpus}', 'w') as f:
    f.write(text)

import pickle
D2B_compound_map = {}
with open('D2B_compound_map.pickle', 'rb') as f:
  D2B_compound_map = pickle.load(f)

with open('D2BA_noising_map.pickle', 'wb') as f:
  pickle.dump(D2BA_noising_map, f)

with open('D2BA_combined_map.pickle', 'wb') as f:
  pickle.dump({**D2B_compound_map, **D2BA_noising_map}, f)
```

3 Extra

[ ]:

```
!gcloud auth application-default login
!gcloud auth application-default set-quota-project resolute-parity-392608
!gcloud auth login
```

```
Go to the following link in your browser:

    https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=76408
6051850-6qr4p6gpi6hn506pt8ejuq83di341hur.apps.googleusercontent.com&redirect_uri
=https%3A%2F%2Fsdk.cloud.google.com%2Fapplicationdefaultauthcode.html&scope=open
id+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.go
ogleapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fs
qlservice.login&state=pn3jj5BdTojkiMLb0rEZFrWLXOSD1p&prompt=consent&access_type=
offline&code_challenge=lxJSODtVjS4qAF4EzPHeZv7J9i-hld5sSNliI7eY3_I&code_challeng
e_method=S256

Enter authorization code:
4/0AfJohXkUDwBTRmWfhgszDOWBMt_iwnTbYGx7ErwsUgHynqk92JLdtjh1CxfT0zY0HWK5IQ

Credentials saved to file:
[/content/.config/application_default_credentials.json]

These credentials will be used by any library that requests Application Default
Credentials (ADC).
WARNING:
Cannot find a quota project to add to ADC. You might receive a "quota exceeded"
or "API not enabled" error. Run $ gcloud auth application-default set-quota-
project to add a quota project.

Credentials saved to file:
[/content/.config/application_default_credentials.json]

These credentials will be used by any library that requests Application Default
Credentials (ADC).

Quota project "resolute-parity-392608" was added to ADC which can be used by
Google client libraries for billing and quota. Note that some services may still
bill the project owning the resource.
Go to the following link in your browser:

    https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=32555
940559.apps.googleusercontent.com&redirect_uri=https%3A%2F%2Fsdk.cloud.google.co
m%2Fauthcode.html&scope=openid+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinf
o.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fw
ww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2Fwww.googleapis.com%2Fau
th%2Fsqlservice.login+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcompute+https%3A
%2F%2Fwww.googleapis.com%2Fauth%2Faccounts.reauth&state=lXHvwJIi8OIIYmdGqICiMwIJ
```

```
TFdZXj&prompt=consent&access_type=offline&code_challenge=WuFhb-
cWrymEt75K_tlsCo0InEiDopwzPXHYpgdw_Lo&code_challenge_method=S256

Enter authorization code: 4/0AfJohXlkFwQN9Phc-
YBmL2UdC5EeJsJs52_HMxz9VRU_TifjWP3hvVP8mddhkJLNWPZYHg

You are now logged in as [luckyblock0@gmail.com].
Your current project is [None].  You can change this setting by running:
  $ gcloud config set project PROJECT_ID
```

[ ]:

```python
import requests
import subprocess
import json

def romanize_text(src, contents):
    # Define the data you want to send in the POST request
    data = {
        'source_language_code': src,
        'contents': contents
    }

    # Define the file path where you want to save the JSON data
    file_path = 'request.json'

    # Write the dictionary to the JSON file
    with open(file_path, 'w') as json_file:
        json.dump(data, json_file, indent=4)

    command = """
    curl -X POST \
        -H "Authorization: Bearer $(gcloud auth print-access-token)" \
        -H "x-goog-user-project: resolute-parity-392608" \
        -H "Content-Type: application/json; charset=utf-8" \
        -d @request.json \
        "https://translation.googleapis.com/v3/projects/resolute-parity-392608/
 ↪locations/us-central1:romanizeText" \
        -o "romanized.json"
    """
    subprocess.run(command, shell=True, text=True, stdout=subprocess.PIPE)

    romanized_text_list = []
    with open('romanized.json', 'r') as f:
      romanized_texts = dict(json.load(f))['romanizations']

      for roman_text in romanized_texts:
        romanized_text_list.append(roman_text['romanizedText'])
    return romanized_text_list

# romanize_text('ja', ['   ', '         '])

def translate_text(target: str, source: str, text: str) -> dict:
    """Translates text into the target language.

    Target must be an ISO 639-1 language code.
    See https://g.co/cloud/translate/v2/translate-reference#supported_languages
    """
    from google.cloud import translate_v2 as translate

    translate_client = translate.Client()

    if isinstance(text, bytes):
        text = text.decode("utf-8")

    result = translate_client.translate(text, target_language=target,␣
 ↪source_language=source)

    non_roman = {
      'ar': 'Arabic',
      'am': 'Amharic',
      'bn': 'Bengali',
      'be': 'Belarusian',
      'hi': 'Hindi',
      'ja': 'Japanese',
      'my': 'Myanmar',
      'ru': 'Russian',
      'sr': 'Serbian',
      'uk': 'Ukrainian'
    }

    if target in non_roman.keys():
      result = romanize_text(target, result['translatedText'])
      return result
    return result['translatedText']
```

[ ]:

```
import pickle
en_words = []
en_dicts = {}

with open('translate_map.pickle', 'rb') as f:
  en_dicts = pickle.load(f)

en_words = list(en_dicts.keys())
```

[ ]:

```
from IPython.core.alias import default_aliases
from tqdm import tqdm

import random

prob = 0.2 # 20% of the dictionary will be cognated
flag_translate = [1] * len(en_words)

i = 0
translate_map = {}
for word in en_words:
  translate_map[word] = flag_translate[i]
  i += 1

C1_mapping = {}
C2_mapping = {}
D1_mapping = {}
D2_mapping = {}

languages = ['de', 'pt', 'af', 'gl']

de_cache = {}
pt_cache = {}
af_cache = {}
gl_cache = {}

def transform_de_map(translate_map, tgt):
  global de_cache
  result_map = {}
  for word, flag in tqdm(translate_map.items()):
    if flag:
      if word in de_cache:
        continue
      else:
        result_map[word] = translate_text(tgt, 'en', word)
        de_cache[word] = result_map[word]
    else:
      result_map[word] = word
      de_cache[word] = result_map[word]
  return result_map

def transform_pt_map(translate_map, tgt):
  global pt_cache
  result_map = {}
  for word, flag in tqdm(translate_map.items()):
    if flag:
      if word in de_cache:
        continue
      else:
        result_map[word] = translate_text(tgt, 'en', word)
        pt_cache[word] = result_map[word]
    else:
      result_map[word] = word
      pt_cache[word] = result_map[word]
  return result_map

def transform_gl_map(translate_map, tgt):
  global gl_cache
  result_map = {}
  for word, flag in tqdm(translate_map.items()):
    if flag:
      if word in gl_cache:
        continue
      else:
        result_map[word] = translate_text(tgt, 'en', word)
        gl_cache[word] = result_map[word]
    else:
      result_map[word] = word
      gl_cache[word] = result_map[word]
  return result_map
```

[ ]:

```
import pickle
from google.colab import files

en_de_map = transform_de_map(translate_map, 'de')
with open('en_de_map.pickle', 'wb') as f:
    pickle.dump(en_de_map, f)
files.download('en_de_map.pickle')
```

```
100%|      | 10608/10608 [2:06:52<00:00,  1.39it/s]

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[ ]:

```
import pickle
from google.colab import files

def transform_pt_map(translate_map, tgt):
  global pt_cache
  result_map = {}
  for word, flag in tqdm(translate_map.items()):
    if flag:
      if word in pt_cache:
        continue
```

```
   else:
    result_map[word] = translate_text(tgt, 'en', word)
    pt_cache[word] = result_map[word]
  else:
   result_map[word] = word
   pt_cache[word] = result_map[word]
 return result_map

en_pt_map = transform_pt_map(translate_map, 'pt')
with open('en_pt_map.pickle', 'wb') as f:
  pickle.dump(en_pt_map, f)
files.download('en_pt_map.pickle')
```

```
100%|     | 10608/10608 [2:07:08<00:00,  1.39it/s]

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[ ]:

```
import pickle
from google.colab import files

def transform_af_map(translate_map, tgt, default = {}):
 i = 1
 result_map = default
 for word, flag in tqdm(translate_map.items()):
  if flag:
   if word in af_cache:
    continue
   else:
    result_map[word] = translate_text(tgt, 'en', word)
    af_cache[word] = result_map[word]
  else:
   result_map[word] = word
   af_cache[word] = result_map[word]

  if (i % 1000) == 0:
   with open(f'en_af_map-{i}.pickle', 'wb') as f:
    pickle.dump(result_map, f)

  i += 1

 return result_map

en_af_map = transform_af_map(translate_map, 'af')
with open('en_af_map.pickle', 'wb') as f:
  pickle.dump(en_af_map, f)
files.download('en_af_map.pickle')
```

```
100%|      | 10608/10608 [16:50<00:00, 10.50it/s]

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[ ]:

```python
with open('en_af_map.pickle', 'wb') as f:
    pickle.dump(af_cache, f)
files.download('en_af_map.pickle')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

[ ]:

```python
len(af_cache)
```

[ ]:

```
10608
```

[ ]:

```python
import pickle
from google.colab import files

def transform_gl_map(translate_map, tgt, default = {}):
  i = 1
  result_map = default
  for word, flag in tqdm(translate_map.items()):
    if flag:
      if word in gl_cache:
        continue
      else:
        result_map[word] = translate_text(tgt, 'en', word)
        gl_cache[word] = result_map[word]
    else:
      result_map[word] = word
      gl_cache[word] = result_map[word]

    if (i % 1000) == 0:
      with open(f'en_gl_map-{i}.pickle', 'wb') as f:
        pickle.dump(result_map, f)
      files.download(f'en_gl_map-{i}.pickle')

    i += 1

  return result_map

en_gl_map = transform_gl_map(translate_map, 'gl')
with open('en_gl_map.pickle', 'wb') as f:
    pickle.dump(en_gl_map, f)
files.download('en_gl_map.pickle')
```