In [1]:
```
import re
import json
import argparse
import os
import sys
import torch
import pandas as pd
import numpy as np

from tqdm import tqdm
from nltk.stem import PorterStemmer
# from transformers import GPT2Tokenizer, GPT2Model
from transformers import AutoTokenizer, AutoModelForCausalLM
from huggingface_hub import login

from swisscom_ai.research_keyphrase.preprocessing.postagging import PosTaggingCoreNLP
from swisscom_ai.research_keyphrase.model.input_representation import InputTextObj
from swisscom_ai.research_keyphrase.model.extractor import extract_candidates
```

```
C:\Users\user0\anaconda3\envs\LLM_ENV\Lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [2]:
```
host = 'localhost'
port = 9000
pos_tagger = PosTaggingCoreNLP(host, port)

# load stopwords
stopwords = []
with open('UGIR_stopwords.txt', "r") as f:
    for line in f:
        if line:
            stopwords.append(line.replace('\n', ''))

stemmer = PorterStemmer()

def read_jsonl(path):
    data = []
    with open(path, 'r') as f:
        for line in f:
            item = json.loads(line.strip())
            data.append(item)
    return data
```

In [3]:
```
def get_candidates(core_nlp, text):
    tagged = core_nlp.pos_tag_raw_text(text)
    text_obj = InputTextObj(tagged, 'en')
    candidates = extract_candidates(text_obj)
    return candidates

def get_phrase_indices(text_tokens, phrase, prefix):
    text_tokens = [t.replace(prefix, '') for t in text_tokens]

    phrase = phrase.replace(' ', '')

    matched_indices = []
    matched_index = []
```

```
    target = phrase
    for i in range(len(text_tokens)):
        cur_token = text_tokens[i]
        sub_len = min(len(cur_token), len(phrase))
        if cur_token[:sub_len].lower() == target[:sub_len]:
            matched_index.append(i)
            target = target[sub_len:]
            if len(target) == 0:
                matched_indices.append([matched_index[0], matched_index[-1] + 1])
                target = phrase
        else:
            matched_index = []
            target = phrase
            if cur_token[:sub_len].lower() == target[:sub_len]:
                matched_index.append(i)
                target = target[sub_len:]
                if len(target) == 0:
                    matched_indices.append([matched_index[0], matched_index[-1] + 1])
                    target = phrase

    return matched_indices

def remove_repeated_sub_word(candidates_pos_dict):
    for phrase in candidates_pos_dict.keys():
        split_phrase = re.split(r'\s+|-', phrase)
        split_phrase = list(filter(None, split_phrase))
        if len(split_phrase) > 1:
            for word in split_phrase:
                if word in candidates_pos_dict:
                    single_word_positions = candidates_pos_dict[word]
                    phrase_positions = candidates_pos_dict[phrase]
                    single_word_alone_positions = [pos for pos in single_word_positions if not a
                        pos[0] >= phrase_pos[0] and pos[1] <= phrase_pos[1] for phrase_pos in ph
                    candidates_pos_dict[word] = single_word_alone_positions

    return candidates_pos_dict

def get_all_indices(candidates_pos_dict, window_end):
    all_indices = []
    for phrase in candidates_pos_dict.values():
        for element in phrase:
            start_index = element[0]
            end_index = element[1]
            if(start_index>window_end):
                all_indices.extend(range(start_index, end_index))
    all_indices = sorted(all_indices)
    all_indices = list(dict.fromkeys(all_indices))

    return all_indices

def aggregate_phrase_scores(index_list, tokens_scores):
    total_score = 0.0

    for p_index in index_list:
        part_sum = tokens_scores[p_index[0]:p_index[1]].sum()
        total_score += part_sum

    return total_score
```

```python
def get_score_full(candidates, references, maxDepth=15):
    precision = []
    recall = []
    reference_set = set(references)
    referencelen = len(reference_set)
    true_positive = 0
    for i in range(maxDepth):
        if len(candidates) > i:
            kp_pred = candidates[i]
            if kp_pred in reference_set:
                true_positive += 1
            precision.append(true_positive / float(i + 1))
            recall.append(true_positive / float(referencelen))
        else:
            precision.append(true_positive / float(len(candidates)))
            recall.append(true_positive / float(referencelen))
    return precision, recall


def evaluate_document(candidates, ground_truth):
    results = {}
    precision_scores, recall_scores, f1_scores = {5: [], 10: [], 15: []}, \
                                                 {5: [], 10: [], 15: []}, \
                                                 {5: [], 10: [], 15: []}
    for candidate, gt in zip(candidates, ground_truth):
        p, r = get_score_full(candidate, gt)
        for i in [5, 10, 15]:
            precision = p[i - 1]
            recall = r[i - 1]
            if precision + recall > 0:
                f1_scores[i].append((2 * (precision * recall)) / (precision + recall))
            else:
                f1_scores[i].append(0)
            precision_scores[i].append(precision)
            recall_scores[i].append(recall)

    print("########################\nMetrics")
    for i in precision_scores:
        print("@{}".format(i))
        print("F1:{}".format(np.mean(f1_scores[i])))
        print("P:{}".format(np.mean(precision_scores[i])))
        print("R:{}".format(np.mean(recall_scores[i])))

        top_n_p = 'precision@' + str(i)
        top_n_r = 'recall@' + str(i)
        top_n_f1 = 'f1@' + str(i)
        results[top_n_p] = np.mean(precision_scores[i])
        results[top_n_r] = np.mean(recall_scores[i])
        results[top_n_f1] = np.mean(f1_scores[i])
    print("#########################")

    return results

def evaluate_dataset(predicted_top, dataset, score_type, dataset_name):
    experiment_results = []
    gt_keyphrase_list = []
    predicted_keyphrase_list = []

    for i in range(len(dataset)):
```

```
        predicted_keyphrase = predicted_top[i]
        predicted_keyphrase = [phrase.lower() for phrase in predicted_keyphrase]
        predicted_keyphrase_list.append(predicted_keyphrase)

        gt_keyphrase = [key.lower() for key in dataset[i]['keyphrases']]
        gt_keyphrase_list.append(gt_keyphrase)

    total_score = evaluate_document(predicted_keyphrase_list, gt_keyphrase_list)
    experiment_results.append(total_score)

    df = pd.DataFrame(experiment_results)

    path = f'experiment_results/{dataset_name}/'
    os.makedirs(path, exist_ok=True)
    df.to_csv(f'{path}score_type_{score_type}.csv', index=False)

    top3_f1_5 = df.nlargest(3, 'f1@5').reset_index(drop=True)
    top3_f1_10 = df.nlargest(3, 'f1@10').reset_index(drop=True)
    top3_f1_15 = df.nlargest(3, 'f1@15').reset_index(drop=True)

    return top3_f1_5, top3_f1_10, top3_f1_15
```

In [4]:
```
login(token=                              )
```

```
The token has not been saved to the git credentials helper. Pass `add_to_git_credential=True` in
this function directly or `--add-to-git-credential` if using via `huggingface-cli` if you want to
set the git credential as well.
Token is valid (permission: read).
Your token has been saved to C:\Users\user0\.cache\huggingface\token
Login successful
```

In [5]:
```
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Meta-Llama-3-8B")
model = AutoModelForCausalLM.from_pretrained("meta-llama/Meta-Llama-3-8B", attn_implementation="
prefix = 'Ġ'
```

```
C:\Users\user0\anaconda3\envs\LLM_ENV\Lib\site-packages\huggingface_hub\file_download.py:1132: Fu
tureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads alwa
ys resume when possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fi
ne-tuned or trained.
Loading checkpoint shards: 100%|██████████| 4/4 [00:12<00:00,  3.08s/it]
C:\Users\user0\anaconda3\envs\LLM_ENV\Lib\site-packages\huggingface_hub\file_download.py:1132: Fu
tureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads alwa
ys resume when possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
```

In [6]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"device: {device}")
```

```
device: cuda
```

DATASET: INSPEC

In [7]:
```
dataset_name = "inspec"
dataset = read_jsonl("KEYWORD_DATA/{}.jsonl".format(dataset_name))
```

SAMRANK BASE

In [8]:

```
model.to(device)
model.eval()

dataset_att_scores_overall = []

for data in tqdm(dataset):
    with torch.no_grad():
        input_text = data["text"]

        tokenized_content = tokenizer(input_text, return_tensors='pt')
        outputs = model(**tokenized_content.to(device))
        content_tokens = tokenizer.convert_ids_to_tokens(tokenized_content['input_ids'].squeeze(

        attentions = outputs.attentions
        del outputs

        candidates = get_candidates(pos_tagger, input_text)
        candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwords]
        candidates_indices = {}
        for phrase in candidates:
            matched_indices = get_phrase_indices(content_tokens, phrase, prefix)
            if len(matched_indices) == 0:
                continue
            candidates_indices[phrase] = matched_indices
        candidates_indices = remove_repeated_sub_word(candidates_indices)

        ######################################################################
        # ATTENTION MEASSUREMENT
        attentions = sum(attentions)/len(attentions)
        attentions = attentions.squeeze(0)
        att_scores = attentions.mean(0).sum(0)
        att_scores[0] = 0
        ######################################################################

        phrase_score_dict = {}
        for phrase in candidates_indices.keys():
            try:
                phrase_indices = candidates_indices[phrase]
                if len(phrase_indices) == 0:
                    continue
            except KeyError:
                continue

            final_phrase_score = aggregate_phrase_scores(phrase_indices, att_scores)

            if len(phrase.split()) == 1:
                final_phrase_score = final_phrase_score / len(phrase_indices)
            phrase_score_dict[phrase] = final_phrase_score

    sorted_scores_att_o_s = sorted(phrase_score_dict.items(), key=lambda item: item[1], reverse=
    stemmed_sorted_scores_att_o_s = [(" ".join(stemmer.stem(word) for word in phrase.split()), s
                                      phrase, score in sorted_scores_att_o_s]

    set_stemmed_scores_list_att_o_s = []
    for phrase, score in stemmed_sorted_scores_att_o_s:
        if phrase not in set_stemmed_scores_list_att_o_s:
            set_stemmed_scores_list_att_o_s.append(phrase)

    pred_stemmed_phrases_att_o_s = set_stemmed_scores_list_att_o_s[:15]
```

```
    dataset_att_scores_overall.append(pred_stemmed_phrases_att_o_s)

att_o_s_top3_f1_5, att_o_s_top3_f1_10, att_o_s_top3_f1_15 = evaluate_dataset(dataset_att_scores_
```

```
100%|██████████| 500/500 [01:55<00:00,  4.33it/s]
########################
Metrics
@5
F1:0.3424774238136697
P:0.4784
R:0.29396447932697767
@10
F1:0.3818415536037738
P:0.38055952380952385
R:0.4318405058729329
@15
F1:0.381147979499465
P:0.32708260073260065
R:0.5150753282616488
########################
```

ATTENTIONSEEKER

In [9]:
```
model.to(device)
model.eval()

dataset_att_scores_overall = []

for data in tqdm(dataset):
    with torch.no_grad():
        input_text = data["text"]

        tokenized_content = tokenizer(input_text, return_tensors='pt')
        outputs = model(**tokenized_content.to(device))
        content_tokens = tokenizer.convert_ids_to_tokens(tokenized_content['input_ids'].squeeze(

        all_attentions = outputs.attentions
        del outputs

        candidates = get_candidates(pos_tagger, input_text)
        candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwords]
        candidates_indices = {}
        for phrase in candidates:
            matched_indices = get_phrase_indices(content_tokens, phrase, prefix)
            if len(matched_indices) == 0:
                continue
            candidates_indices[phrase] = matched_indices
        candidates_indices = remove_repeated_sub_word(candidates_indices)
        all_indices = get_all_indices(candidates_indices,0)

        ###################################################################
        len_t_tokens = all_attentions[0].squeeze(0)[0].shape[0]
        all_indices_tensor = torch.arange(len_t_tokens)
        mask = torch.isin(all_indices_tensor, torch.tensor(all_indices)).to(device)
        mask_1 = mask*1.0
        # ATTENTION MEASSUREMENT
        attentions = torch.zeros(len_t_tokens).to(device)
        for layer in range(len(all_attentions)):
            for head in range(32):
```

```
                crrn_att_map = all_attentions[layer].squeeze(0)[head].clone()
                crrn_att_map[~mask] = 0
                lh_weight = torch.matmul(crrn_att_map,mask_1)
                attentions += lh_weight.mean(0)*torch.matmul(lh_weight,crrn_att_map)
        att_scores = attentions
        att_scores[0] = 0
        # ATTENTION SEEKER (LHC-SEEKER)
        f_att_scores = torch.zeros_like(att_scores)
        f_att_scores[mask] = att_scores[mask]
        # NEW ATTENTION MEASSUREMENT
        attentions = torch.zeros(len_t_tokens).to(device)
        for layer in range(len(all_attentions)):
            for head in range(32):
                crrn_att_map = all_attentions[layer].squeeze(0)[head].clone()
                crrn_att_map[~mask] = 0
                lh_weight = torch.matmul(crrn_att_map,f_att_scores)
                attentions += lh_weight.mean(0)*torch.matmul(lh_weight,crrn_att_map)
        att_scores = attentions
        att_scores[0] = 0
        ###################################################################

        phrase_score_dict = {}
        for phrase in candidates_indices.keys():
            try:
                phrase_indices = candidates_indices[phrase]
                if len(phrase_indices) == 0:
                    continue
            except KeyError:
                continue

            final_phrase_score = aggregate_phrase_scores(phrase_indices, att_scores)

            if len(phrase.split()) == 1:
                final_phrase_score = final_phrase_score / len(phrase_indices)

            phrase_score_dict[phrase] = final_phrase_score

    sorted_scores_att_o_s = sorted(phrase_score_dict.items(), key=lambda item: item[1], reverse=
    stemmed_sorted_scores_att_o_s = [(" ".join(stemmer.stem(word) for word in phrase.split()), s
                                phrase, score in sorted_scores_att_o_s]

    set_stemmed_scores_list_att_o_s = []
    for phrase, score in stemmed_sorted_scores_att_o_s:
        if phrase not in set_stemmed_scores_list_att_o_s:
            set_stemmed_scores_list_att_o_s.append(phrase)

    pred_stemmed_phrases_att_o_s = set_stemmed_scores_list_att_o_s[:15]
    dataset_att_scores_overall.append(pred_stemmed_phrases_att_o_s)

att_o_s_top3_f1_5, att_o_s_top3_f1_10, att_o_s_top3_f1_15 = evaluate_dataset(dataset_att_scores_
```

```
100%|██████████| 500/500 [05:44<00:00,  1.45it/s]
```

```
########################
Metrics
@5
F1:0.35490195392637314
P:0.49440000000000006
R:0.30500898385626474
@10
F1:0.40136623714102004
P:0.3987595238095238
R:0.45497844248364167
@15
F1:0.39220319615918514
P:0.33681593406593413
R:0.5299187073369938
#########################
```

DATASET: SEMEVAL 2017

In [11]:
```
dataset_name = "semeval2017"
dataset = read_jsonl("KEYWORD_DATA/{}.jsonl".format(dataset_name))
```

SAMRANK BASE

In [12]:
```
model.to(device)
model.eval()

dataset_att_scores_overall = []

for data in tqdm(dataset):
    with torch.no_grad():
        input_text = data["text"]

        tokenized_content = tokenizer(input_text, return_tensors='pt')
        outputs = model(**tokenized_content.to(device))
        content_tokens = tokenizer.convert_ids_to_tokens(tokenized_content['input_ids'].squeeze(

        attentions = outputs.attentions
        del outputs

        candidates = get_candidates(pos_tagger, input_text)
        candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwords]
        candidates_indices = {}
        for phrase in candidates:
            matched_indices = get_phrase_indices(content_tokens, phrase, prefix)
            if len(matched_indices) == 0:
                continue
            candidates_indices[phrase] = matched_indices
        candidates_indices = remove_repeated_sub_word(candidates_indices)

        ###################################################################
        # ATTENTION MEASSUREMENT
        attentions = sum(attentions)/len(attentions)
        attentions = attentions.squeeze(0)
        att_scores = attentions.mean(0).sum(0)
        att_scores[0] = 0
        ###################################################################
```

```
        phrase_score_dict = {}
        for phrase in candidates_indices.keys():
            try:
                phrase_indices = candidates_indices[phrase]
                if len(phrase_indices) == 0:
                    continue
            except KeyError:
                continue

            final_phrase_score = aggregate_phrase_scores(phrase_indices, att_scores)

            if len(phrase.split()) == 1:
                final_phrase_score = final_phrase_score / len(phrase_indices)
            phrase_score_dict[phrase] = final_phrase_score

    sorted_scores_att_o_s = sorted(phrase_score_dict.items(), key=lambda item: item[1], reverse=
    stemmed_sorted_scores_att_o_s = [(" ".join(stemmer.stem(word) for word in phrase.split()), s
                                      phrase, score in sorted_scores_att_o_s]

    set_stemmed_scores_list_att_o_s = []
    for phrase, score in stemmed_sorted_scores_att_o_s:
        if phrase not in set_stemmed_scores_list_att_o_s:
            set_stemmed_scores_list_att_o_s.append(phrase)

    pred_stemmed_phrases_att_o_s = set_stemmed_scores_list_att_o_s[:15]
    dataset_att_scores_overall.append(pred_stemmed_phrases_att_o_s)

att_o_s_top3_f1_5, att_o_s_top3_f1_10, att_o_s_top3_f1_15 = evaluate_dataset(dataset_att_scores_
```

```
100%|██████████| 493/493 [02:36<00:00,  3.15it/s]
#########################
Metrics
@5
F1:0.24742282955536773
P:0.5168356997971603
R:0.1700399499230244
@10
F1:0.3350504558633974
P:0.4480730223123732
R:0.2851300385057317
@15
F1:0.3701308557000764
P:0.3989408495493688
R:0.3718292969857578
#########################
```

ATTENTIONSEEKER

In [13]:

```
model.to(device)
model.eval()

dataset_att_scores_overall = []

for data in tqdm(dataset):
    with torch.no_grad():
        input_text = data["text"]

        tokenized_content = tokenizer(input_text, return_tensors='pt')
        outputs = model(**tokenized_content.to(device))
```

```
        content_tokens = tokenizer.convert_ids_to_tokens(tokenized_content['input_ids'].squeeze(

        all_attentions = outputs.attentions
        del outputs

        candidates = get_candidates(pos_tagger, input_text)
        candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwords]
        candidates_indices = {}
        for phrase in candidates:
            matched_indices = get_phrase_indices(content_tokens, phrase, prefix)
            if len(matched_indices) == 0:
                continue
            candidates_indices[phrase] = matched_indices
        candidates_indices = remove_repeated_sub_word(candidates_indices)
        all_indices = get_all_indices(candidates_indices,0)

        ###################################################################
        len_t_tokens = all_attentions[0].squeeze(0)[0].shape[0]
        all_indices_tensor = torch.arange(len_t_tokens)
        mask = torch.isin(all_indices_tensor, torch.tensor(all_indices)).to(device)
        mask_1 = mask*1.0
        # ATTENTION MEASSUREMENT
        attentions = torch.zeros(len_t_tokens).to(device)
        for layer in range(len(all_attentions)):
            for head in range(32):
                crrn_att_map = all_attentions[layer].squeeze(0)[head].clone()
                crrn_att_map[~mask] = 0
                lh_weight = torch.matmul(crrn_att_map,mask_1)
                attentions += lh_weight.mean(0)*torch.matmul(lh_weight,crrn_att_map)
        att_scores = attentions
        att_scores[0] = 0
        # ATTENTION SEEKER (LHC-SEEKER)
        f_att_scores = torch.zeros_like(att_scores)
        f_att_scores[mask] = att_scores[mask]
        # NEW ATTENTION MEASSUREMENT
        attentions = torch.zeros(len_t_tokens).to(device)
        for layer in range(len(all_attentions)):
            for head in range(32):
                crrn_att_map = all_attentions[layer].squeeze(0)[head].clone()
                crrn_att_map[~mask] = 0
                lh_weight = torch.matmul(crrn_att_map,f_att_scores)
                attentions += lh_weight.mean(0)*torch.matmul(lh_weight,crrn_att_map)
        att_scores = attentions
        att_scores[0] = 0
        ###################################################################

        phrase_score_dict = {}
        for phrase in candidates_indices.keys():
            try:
                phrase_indices = candidates_indices[phrase]
                if len(phrase_indices) == 0:
                    continue
            except KeyError:
                continue

            final_phrase_score = aggregate_phrase_scores(phrase_indices, att_scores)

            if len(phrase.split()) == 1:
                final_phrase_score = final_phrase_score / len(phrase_indices)
```

```
            phrase_score_dict[phrase] = final_phrase_score

    sorted_scores_att_o_s = sorted(phrase_score_dict.items(), key=lambda item: item[1], reverse=
    stemmed_sorted_scores_att_o_s = [(" ".join(stemmer.stem(word) for word in phrase.split()), s
                                phrase, score in sorted_scores_att_o_s]

    set_stemmed_scores_list_att_o_s = []
    for phrase, score in stemmed_sorted_scores_att_o_s:
        if phrase not in set_stemmed_scores_list_att_o_s:
            set_stemmed_scores_list_att_o_s.append(phrase)

    pred_stemmed_phrases_att_o_s = set_stemmed_scores_list_att_o_s[:15]
    dataset_att_scores_overall.append(pred_stemmed_phrases_att_o_s)

att_o_s_top3_f1_5, att_o_s_top3_f1_10, att_o_s_top3_f1_15 = evaluate_dataset(dataset_att_scores_
```

```
100%|██████████| 493/493 [06:20<00:00,  1.30it/s]
#########################
Metrics
@5
F1:0.25397632889673977
P:0.5330628803245435
R:0.1741208725412833
@10
F1:0.345326367943574
P:0.46186612576064906
R:0.2942509747499267
@15
F1:0.3849912207748741
P:0.4150328035723573
R:0.38667026905240437
#########################
```

In [ ]: