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)

        stemmed_gt_keyphrases = [" ".join(stemmer.stem(word) for word in phrase.split()) for phr
        gt_keyphrase = [key.lower() for key in stemmed_gt_keyphrases]
        gt_keyphrase_f = list(dict.fromkeys(gt_keyphrase))
        gt_keyphrase_list.append(gt_keyphrase_f)

    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

def get_same_len_segments(total_tokens_ids, max_len):
    num_of_seg = (len(total_tokens_ids) // max_len) + 1
    seg_len = int(len(total_tokens_ids) / num_of_seg)
    segments = []
    attn_masks = []
    for _ in range(num_of_seg):
        if len(total_tokens_ids) > seg_len:
            segment = total_tokens_ids[:seg_len]
            total_tokens_ids = total_tokens_ids[seg_len:]
        else:
            segment = total_tokens_ids
        segments.append(segment)
        attn_masks.append([1] * len(segments[-1]))

    return segments, attn_masks


def clean_data(text,min_length=10):
    pattern_long_numbers = re.compile(r'\b[0-9.,\s]{'+str(min_length)+r',}\b')
    aaa_pattern = r'(AAAA\s*)+'

    text = re.sub(aaa_pattern, '', text)
    text = re.sub(r' -LSB-\s*\d+\s*-RSB-', '', text)
    text = re.sub(r' -LSB-\s*(\d+\s*,\s*){1,7}\d+\s*-RSB-', '', text)
    text = re.sub(r'-LRB- ', '(', text)
    text = re.sub(r' -RRB-', ')', text)
    cleaned_text = re.sub(pattern_long_numbers, '', text)
    cleaned_text = re.sub(r'\s+', ' ', cleaned_text).strip()

    return cleaned_text
```

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: FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads always 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 fine-tuned or trained.
Loading checkpoint shards: 100%|██████████| 4/4 [00:10<00:00,  2.56s/it]
c:\Users\user0\anaconda3\envs\LLM_ENV\Lib\site-packages\huggingface_hub\file_download.py:1132: FutureWarning: `resume_download` is deprecated and will be removed in version 1.0.0. Downloads always 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: SEMEVAL 2010

In [8]:
```
dataset_name = "semeval_test"
dataset = read_jsonl("KEYWORD_DATA/{}.json".format(dataset_name))
```

SAMRANK BASE

In [10]:
```
model.to(device)
model.eval()

max_len = 1024

dataset_att_scores_overall = []

for data in tqdm(dataset):
    with torch.no_grad():
        input_text = data["title"] + ". " + data["abstract"] + " " + clean_data(data["fulltext"]
        tokenized_content = tokenizer(input_text, return_tensors='pt')

        candidates = get_candidates(pos_tagger, input_text)
        candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwords]

        total_tokens_ids = tokenized_content['input_ids'].squeeze(0).tolist()
        total_tokens_ids = total_tokens_ids[1:]

        windows, attention_masks = get_same_len_segments(total_tokens_ids, max_len)

        phrase_score_dict = {}

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

```
            window = windows[i]
            attention_mask = attention_masks[i]
            window = [128000] + window
            attention_mask = [1] + attention_mask
            window = torch.tensor([window])
            attention_mask = torch.tensor([attention_mask])

            outputs = model(window.to(device), attention_mask=attention_mask.to(device))
            attentions = outputs.attentions
            del outputs

            content_tokens = tokenizer.convert_ids_to_tokens(window[0])

            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)

            ###################################################################
            # ATTENTION MEASSUREMENT
            final_attention_map = sum(attentions)/len(attentions)
            final_attention_map = final_attention_map.squeeze(0)
            del attentions
            ######################################################
            att_scores = final_attention_map.mean(0).sum(0)
            ###################################################################
            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)

                if phrase not in phrase_score_dict:
                    phrase_score_dict[phrase] = 0

                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%|██████████| 100/100 [19:58<00:00, 11.99s/it]
########################
Metrics
@5
F1:0.17862037678069187
P:0.35
R:0.12116366512172323
@10
F1:0.20991795846392752
P:0.26099999999999995
R:0.1787202349113545
@15
F1:0.2207351660739788
P:0.21999999999999997
R:0.22647013805720015
#########################
```

ATTENTIONSEEKER

In [12]:

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

max_len = 1024

dataset_att_scores_overall = []

for data in tqdm(dataset):
    with torch.no_grad():
        phrase_score_dict = {}

        first_input_text = data["title"] + ". " + data["abstract"]

        tokenized_content = tokenizer(first_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_abs = get_candidates(pos_tagger, first_input_text)
        candidates_abs = [phrase for phrase in candidates_abs if phrase.split(' ')[0] not in sto
        candidates_indices = {}
        for phrase in candidates_abs:
            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_abs_indices = get_all_indices(candidates_indices,0)

        ################################################################
        # ABSTRACT STRENGHT MASK
        len_t_tokens = all_attentions[0].squeeze(0)[0].shape[0]
        abs_indices_tensor = torch.arange(len_t_tokens)
        mask_abs = torch.isin(abs_indices_tensor, torch.tensor(all_abs_indices)).to(device)*1.0
        ################################################################
```

```
        # ATTENTION MEASSUREMENT
        attentions = torch.zeros(len_t_tokens,len_t_tokens).to(device)
        all_weights = torch.zeros(1024)
        for layer in range(len(all_attentions)):
            for head in range(32):
                lh_weight = torch.matmul(all_attentions[layer].squeeze(0)[head],mask_abs).mean(0
                all_weights[32*layer+head] = lh_weight
                attentions += lh_weight*all_attentions[layer].squeeze(0)[head]
        attentions = attentions/all_weights.mean() #average
        att_scores = attentions.sum(0)
        att_scores[0] = 0
        ###################################################################
        # ABSTRACT STRENGHT
        abs_weight = torch.dot(att_scores,att_scores*mask_abs)
        ###################################################################

        abs_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] = abs_weight*final_phrase_score
            abs_dict[phrase] = final_phrase_score

        ###################################################################
        input_text = clean_data(data["fulltext"])
        tokenized_content = tokenizer(input_text, return_tensors='pt')

        candidates = get_candidates(pos_tagger, input_text)
        candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwords]

        total_tokens_ids = tokenized_content['input_ids'].squeeze(0).tolist()
        total_tokens_ids = total_tokens_ids[1:]

        windows, attention_masks = get_same_len_segments(total_tokens_ids, max_len)

        for i in range(len(windows)):
            window = windows[i]
            attention_mask = attention_masks[i]
            window = [128000] + window
            attention_mask = [1] + attention_mask
            window = torch.tensor([window])
            attention_mask = torch.tensor([attention_mask])

            outputs = model(window.to(device), attention_mask=attention_mask.to(device))
            all_attentions = outputs.attentions
            del outputs

            content_tokens = tokenizer.convert_ids_to_tokens(window[0])
```

```
            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)
            ####################################################################
            # ABSTRACT STRENGHT MASK
            len_t_tokens = all_attentions[0].squeeze(0)[0].shape[0]
            mask_abs = torch.zeros(len_t_tokens).to(device)
            ####################################################################
            abs_candidates_indices = {}
            for phrase in abs_dict.keys():
                matched_indices = get_phrase_indices(content_tokens, phrase, prefix)
                if len(matched_indices) == 0:
                    continue
                abs_candidates_indices[phrase] = matched_indices
            abs_candidates_indices = remove_repeated_sub_word(abs_candidates_indices)
            for phrase, phrase_idx in abs_candidates_indices.items():
                n_ocurrences = len(phrase_idx)
                for p_idx in phrase_idx:
                    mask_abs[p_idx[0]:p_idx[1]] = abs_dict[phrase]/n_ocurrences

            ####################################################################
            all_indices_tensor = torch.arange(len_t_tokens)
            mask_1 = torch.isin(all_indices_tensor, torch.tensor(all_indices)).to(device)*1.0
            # ATTENTION MEASSUREMENT
            attentions = torch.zeros(len_t_tokens,len_t_tokens).to(device)
            all_weights = torch.zeros(1024)
            for layer in range(len(all_attentions)):
                for head in range(32):
                    lh_weight = torch.matmul(all_attentions[layer].squeeze(0)[head],mask_1).mean
                    all_weights[32*layer+head] = lh_weight
                    attentions += lh_weight*all_attentions[layer].squeeze(0)[head]
            attentions = attentions/all_weights.mean() #average
            att_scores = attentions.sum(0)
            att_scores[0] = 0
            ####################################################################
            # ABSTRACT STRENGHT
            abs_weight = torch.dot(att_scores,mask_abs)
            ####################################################################
            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)

                if phrase not in phrase_score_dict:
                    phrase_score_dict[phrase] = 0
```

```
                phrase_score_dict[phrase] += abs_weight*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%|██████████| 100/100 [23:20<00:00, 14.00s/it]
#########################
Metrics
@5
F1:0.1900182168077821
P:0.37200000000000005
R:0.1289899411848414
@10
F1:0.23067822250889616
P:0.285
R:0.19731695203550376
@15
F1:0.2381115218754553
P:0.23866666666666664
R:0.24339174591744903
#########################
```

DATASET: KRAPIVIN

In [15]:
```
dataset_name = "krapivin_test"
dataset = read_jsonl("KEYWORD_DATA/{}.json".format(dataset_name))
```

SAMRANK BASE

In [16]:
```
model.to(device)
model.eval()

max_len = 1024

dataset_att_scores_overall = []

for data in tqdm(dataset):
    with torch.no_grad():
        input_text = data["title"] + " " + data["abstract"] + " " + clean_data(data["fulltext"])
        tokenized_content = tokenizer(input_text, return_tensors='pt')

        try:
            candidates = get_candidates(pos_tagger, input_text)
            candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwor
        except:
            input_len = len(input_text)
```

```
        candidates = get_candidates(pos_tagger, input_text[:int(input_len/2)])
        candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwor
        candidates_2 = get_candidates(pos_tagger, input_text[int(input_len/2):])
        candidates_2 = [phrase for phrase in candidates_2 if phrase.split(' ')[0] not in sto
        candidates.extend(candidates_2)

    total_tokens_ids = tokenized_content['input_ids'].squeeze(0).tolist()
    total_tokens_ids = total_tokens_ids[1:]

    windows, attention_masks = get_same_len_segments(total_tokens_ids, max_len)

    phrase_score_dict = {}

    for i in range(len(windows)):
        window = windows[i]
        attention_mask = attention_masks[i]
        window = [128000] + window
        attention_mask = [1] + attention_mask
        window = torch.tensor([window])
        attention_mask = torch.tensor([attention_mask])

        outputs = model(window.to(device), attention_mask=attention_mask.to(device))
        attentions = outputs.attentions
        del outputs

        content_tokens = tokenizer.convert_ids_to_tokens(window[0])

        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)

        ###################################################################
        # ATTENTION MEASSUREMENT
        final_attention_map = sum(attentions)/len(attentions)
        final_attention_map = final_attention_map.squeeze(0)
        del attentions
        ###################################################
        att_scores = final_attention_map.mean(0).sum(0)
        ###################################################################
        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)

            if phrase not in phrase_score_dict:
                phrase_score_dict[phrase] = 0
```

```
                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%|██████████| 460/460 [1:32:25<00:00, 12.06s/it]
#########################
Metrics
@5
F1:0.17380708525347047
P:0.18130434782608698
R:0.18548639498952463
@10
F1:0.16775853685443953
P:0.13260869565217392
R:0.26677021852491295
@15
F1:0.15151103031923596
P:0.106231884057971
R:0.3165032978941311
#########################
```

ATTENTIONSEEKER

In [17]:

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

max_len = 1024

dataset_att_scores_overall = []

for data in tqdm(dataset):
    with torch.no_grad():
        phrase_score_dict = {}

        first_input_text = data["title"] + " " + data["abstract"]

        tokenized_content = tokenizer(first_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_abs = get_candidates(pos_tagger, first_input_text)
        candidates_abs = [phrase for phrase in candidates_abs if phrase.split(' ')[0] not in sto
        candidates_indices = {}
```

```
        for phrase in candidates_abs:
            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_abs_indices = get_all_indices(candidates_indices,0)

        ####################################################################
        # ABSTRACT STRENGHT MASK
        len_t_tokens = all_attentions[0].squeeze(0)[0].shape[0]
        abs_indices_tensor = torch.arange(len_t_tokens)
        mask_abs = torch.isin(abs_indices_tensor, torch.tensor(all_abs_indices)).to(device)*1.0
        ####################################################################
        # ATTENTION MEASSUREMENT
        attentions = torch.zeros(len_t_tokens,len_t_tokens).to(device)
        all_weights = torch.zeros(1024)
        for layer in range(len(all_attentions)):
            for head in range(32):
                lh_weight = torch.matmul(all_attentions[layer].squeeze(0)[head],mask_abs).mean(0
                all_weights[32*layer+head] = lh_weight
                attentions += lh_weight*all_attentions[layer].squeeze(0)[head]
        attentions = attentions/all_weights.mean() #average
        att_scores = attentions.sum(0)
        att_scores[0] = 0
        ####################################################################
        # ABSTRACT STRENGHT
        abs_weight = torch.dot(att_scores,att_scores*mask_abs)
        ####################################################################

        abs_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] = abs_weight*final_phrase_score
            abs_dict[phrase] = final_phrase_score

        ####################################################################
        input_text = clean_data(data["fulltext"])
        tokenized_content = tokenizer(input_text, return_tensors='pt')

        try:
            candidates = get_candidates(pos_tagger, input_text)
            candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwor
        except:
            input_len = len(input_text)
            candidates = get_candidates(pos_tagger, input_text[:int(input_len/2)])
            candidates = [phrase for phrase in candidates if phrase.split(' ')[0] not in stopwor
            candidates_2 = get_candidates(pos_tagger, input_text[int(input_len/2):])
```

```
            candidates_2 = [phrase for phrase in candidates_2 if phrase.split(' ')[0] not in sto
            candidates.extend(candidates_2)

        total_tokens_ids = tokenized_content['input_ids'].squeeze(0).tolist()
        total_tokens_ids = total_tokens_ids[1:]

        windows, attention_masks = get_same_len_segments(total_tokens_ids, max_len)

        for i in range(len(windows)):
            window = windows[i]
            attention_mask = attention_masks[i]
            window = [128000] + window
            attention_mask = [1] + attention_mask
            window = torch.tensor([window])
            attention_mask = torch.tensor([attention_mask])

            outputs = model(window.to(device), attention_mask=attention_mask.to(device))
            all_attentions = outputs.attentions
            del outputs

            content_tokens = tokenizer.convert_ids_to_tokens(window[0])

            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)
            ##################################################################
            # ABSTRACT STRENGHT MASK
            len_t_tokens = all_attentions[0].squeeze(0)[0].shape[0]
            mask_abs = torch.zeros(len_t_tokens).to(device)
            ##################################################################
            abs_candidates_indices = {}
            for phrase in abs_dict.keys():
                matched_indices = get_phrase_indices(content_tokens, phrase, prefix)
                if len(matched_indices) == 0:
                    continue
                abs_candidates_indices[phrase] = matched_indices
            abs_candidates_indices = remove_repeated_sub_word(abs_candidates_indices)
            for phrase, phrase_idx in abs_candidates_indices.items():
                n_ocurrences = len(phrase_idx)
                for p_idx in phrase_idx:
                    mask_abs[p_idx[0]:p_idx[1]] = abs_dict[phrase]/n_ocurrences

            ##################################################################
            all_indices_tensor = torch.arange(len_t_tokens)
            mask_1 = torch.isin(all_indices_tensor, torch.tensor(all_indices)).to(device)*1.0
            # ATTENTION MEASSUREMENT
            attentions = torch.zeros(len_t_tokens,len_t_tokens).to(device)
            all_weights = torch.zeros(1024)
            for layer in range(len(all_attentions)):
                for head in range(32):
                    lh_weight = torch.matmul(all_attentions[layer].squeeze(0)[head],mask_1).mean
                    all_weights[32*layer+head] = lh_weight
                    attentions += lh_weight*all_attentions[layer].squeeze(0)[head]
            attentions = attentions/all_weights.mean() #average
```

```
            att_scores = attentions.sum(0)
            att_scores[0] = 0
            ###################################################################
            # ABSTRACT STRENGHT
            abs_weight = torch.dot(att_scores,mask_abs)
            ###################################################################
            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)

                if phrase not in phrase_score_dict:
                    phrase_score_dict[phrase] = 0

                phrase_score_dict[phrase] += abs_weight*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%|██████████| 460/460 [1:47:24<00:00, 14.01s/it]
#########################
Metrics
@5
F1:0.2079049867447276
P:0.21695652173913044
R:0.22533917404180898
@10
F1:0.18250253406400352
P:0.14413043478260873
R:0.29022805296100823
@15
F1:0.16220369229617648
P:0.11391304347826085
R:0.3389678464399154
#########################
```