Note - random split of evaluation and training tasks determined at download, divided as follows as a static option: Evaluation:

1. subtask002_quoref_answer_generation --Answer Generation(AG)
2. subtask003_mctaco_question_generation_event_duration -- Question Generation(QG)
3. subtask005_mctaco_wrong_answer_generation_event_duration -- Incorrect Answer Generation(IAG)
4. subtask008_mctaco_wrong_answer_generation_transient_stationary -- IAG
5. subtask022_cosmosqa_passage_inappropriate_binary -- Classification(CF)
6. subtask033_winogrande_answer_generation -- AG
7. subtask034_winogrande_question_modification_object -- Minimal Text Modification(MM)
8. subtask039_qasc_find_overlapping_words -- Verification(VF)
9. subtask040_qasc_question_generation -- QG
10. subtask044_essential_terms_identifying_essential_words -- VF
11. subtask045_miscellaneous_sentence_paraphrasing -- MM
12. subtask052_multirc_identify_bad_question -- CF Currently randomly generates a subset

In [6]:

```
import random
import copy
# Random Split Generation - SOME FILE NAMES ARE WRONG THIS NEEDS TO BE FIXED
categories = {'QG': ['subtask001_quoref_question_generation',
                     'subtask003_mctaco_question_generation_event_duration',
                     'subtask006_mctaco_question_generation_transient_stationary'
                     'subtask009_mctaco_question_generation_event_ordering',
                     'subtask012_mctaco_question_generation_absolute_timepoint',
                     'subtask015_mctaco_question_generation_frequency',
                     'subtask023_cosmosqa_question_generation',
                     'subtask026_drop_question_generation',
                     'subtask031_winogrande_question_generation_object',
                     'subtask032_winogrande_question_generation_person',
                     'subtask040_qasc_question_generation',
                     'subtask048_multirc_question_generation',
                     'subtask060_ropes_question_generation4'],
              'AG': ['subtask002_quoref_answer_generation',
                     'subtask004_mctaco_answer_generation_event_duration',
                     'subtask007_mctaco_answer_generation_transient_stationary',
                     'subtask010_mctaco_answer_generation_event_ordering',
                     'subtask013_mctaco_answer_generation_absolute_timepoint',
                     'subtask016_mctaco_answer_generation_frequency',
                     'subtask024_cosmosqa_answer_generation',
                     'subtask028_drop_answer_generation',
                     'subtask033_winogrande_answer_generation',
                     'subtask041_qasc_answer_generation',
                     'subtask043_essential_terms_answering_incomplete_questions',
                     'subtask047_miscellaenous_answering_science_questions',
                     'subtask051_multirc_correct_answer_single_sentence',
                     'subtask054_multirc_write_correct_answer',
                     'subtask058_multirc_question_answering',
                     'subtask061_ropes_answer_generation4'],
              'IAG': ['subtask005_mctaco_wrong_answer_generation_event_duration',
                      'subtask008_mctaco_wrong_answer_generation_transient_static
                      'subtask011_mctaco_wrong_answer_generation_event_ordering',
                      'subtask014_mctaco_wrong_answer_generation_absolute_timepoi
                      'subtask017_mctaco_wrong_answer_generation_frequency',
                      'subtask025_cosmosqa_incorrect_answer_generation',
                      'subtask042_qasc_incorrect_option_generation',
                      'subtask055_multirc_write_incorrect_answer'],
              'CF': ['subtask018_mctaco_temporal_reasoning_presence',
                     'subtask019_mctaco_temporal_reasoning_category',
                     'subtask020_mctaco_span_based_question',
                     'subtask021_mctaco_grammatical_logical',
                     'subtask022_cosmosqa_passage_inappropriate_binary',
                     'subtask027_drop_answer_type_generation',
                     'subtask046_miscellaenous_question_typing',
                     'subtask049_multirc_questions_needed_to_answer',
                     'subtask050_multirc_answerability',
                     'subtask052_multirc_identify_bad_question',
                     'subtask056_multirc_classify_correct_answer',
                     'subtask057_multirc_classify_incorrect_answer',
                     ],
              'MM': ['subtask029_winogrande_full_object',
                     'subtask030_winogrande_full_person',
                     'subtask034_winogrande_question_modification_object',
```

```
                    'subtask035_winogrande_question_modification_person',
                    'subtask036_qasc_topic_word_to_generate_related_fact',
                    'subtask037_qasc_generate_related_fact',
                    'subtask038_qasc_combined_fact',
                    'subtask045_miscellaneous_sentence_paraphrasing',
                    'subtask053_multirc_correct_bad_question',
                    'subtask059_ropes_story_generation4'],
             'VF': ['subtask039_qasc_find_overlapping_words',
                    'subtask044_essential_terms_identifying_essential_words',
                    ],
             }

# Move two random subtasks from each category into the evaluation subtasks
trainingPrompts = copy.deepcopy(categories)
evaluationPrompts = {'QG': [], 'AG': [], 'IAG': [], 'CF': [], 'MM': [], 'VF': []}
for key in trainingPrompts.keys():
    subtask = random.choice(trainingPrompts[key])
    trainingPrompts[key].remove(subtask)
    evaluationPrompts[key].append(subtask)
    subtask = random.choice(trainingPrompts[key])
    trainingPrompts[key].remove(subtask)
    evaluationPrompts[key].append(subtask)
```

In [8]:
```
# Run this block after to use preset subtasks, do not run to use random subtasks
trainingPrompts = copy.deepcopy(categories)
evaluationPrompts = {'QG': ['subtask003_mctaco_question_generation_event_duration
                            'subtask040_qasc_question_generation'],
                     'AG': ['subtask002_quoref_answer_generation',
                            'subtask033_winogrande_answer_generation'],
                     'IAG': ['subtask005_mctaco_wrong_answer_generation_event_dur
                             'subtask008_mctaco_wrong_answer_generation_transient
                     'CF': ['subtask022_cosmosqa_passage_inappropriate_binary',
                            'subtask052_multirc_identify_bad_question'],
                     'MM': ['subtask034_winogrande_question_modification_object',
                            'subtask045_miscellaneous_sentence_paraphrasing'],
                     'VF': ['subtask039_qasc_find_overlapping_words',
                            'subtask044_essential_terms_identifying_essential_wor

for key in trainingPrompts.keys():
    for subtask in evaluationPrompts[key]:
        trainingPrompts[key].remove(subtask)
```

In [3]:
```
# Models
from transformers import BartTokenizer, BartModel, GPT2Tokenizer, GPT2Model


random_number_model = (lambda **x: random.choice(['One', 'Two', 'Three', 'Four',
random_tokenizer = BartTokenizer.from_pretrained('facebook/bart-base') #just usir

# Needs to be pretrained, takes a long time when untrained(might also be bad code
bart_model = BartModel.from_pretrained('facebook/bart-base')
bart_tokenizer = BartTokenizer.from_pretrained('facebook/bart-base')

# For testing a slightly better baseline than random, should look at GPT3
gpt2_tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
gpt2_model = GPT2Model.from_pretrained('gpt2')
```

In [4]:
```
# Instructions Encoding - add pos/neg examples later

def no_examples_encoding(task, inp):
    return f"""
Definition: {task['Definition']}
Prompt: {task['Prompt']}
Things to Avoid: {task['Things to Avoid']}
Emphasis&Caution: {task['Emphasis & Caution']}
Input: {inp}
Output:
"""
```

In [8]:
```
# Evaluation(currently for models trained on no_examples_encoding)
import numpy as np
import json
from torchmetrics.text.rouge import ROUGEScore

models = {'Random': random_number_model, 'Bart': bart_model}
tokenizers = {'Random': random_tokenizer, 'Bart': bart_tokenizer}
# models = {'Random': random_number_model, 'GPT2': gpt2_model}
# tokenizers = {'Random': random_tokenizer, 'GPT2': gpt2_tokenizer}
scores = {}
scorer = ROUGEScore(rouge_keys=('rougeL'))

for category in evaluationPrompts.keys():
    scoresForCategory = {}
    for model in models.keys():
        scoresForCategory[model] = []
    for task in evaluationPrompts[category]:
        with open('./app_static_tasks_sample/' + task + '.json') as json_file:
            subtask = json.load(json_file)
            for instance in subtask['Instances']:
                string_encoding = no_examples_encoding(subtask, instance['input']
                for model in models.keys():
                    tokenizer = tokenizers[model]
                    inputs = tokenizer(string_encoding, return_tensors="pt") # th
                    if (len(inputs) >= 1024):
                        print('ignored prompt')
                        continue #temporary fix to avoid prompts which are too lo
                    try: #FIX THIS LATER
                        outputs = models[model](**inputs)
                        rgeScores = scorer(instance['output'], outputs)
                        scoresForCategory[model].append(rgeScores['rougeL_fmeasur
                    except:
                        continue
    for model in models.keys():
        scoresForCategory[model] = sum(scoresForCategory[model]) / len(scoresForC
    scores[category] = scoresForCategory
    print(category + " complete")

print(scores)
```

```
QG complete

Token indices sequence length is longer than the specified maximum sequence len
gth for this model (1242 > 1024). Running this sequence through the model will
result in indexing errors

AG complete
IAG complete
CF complete
MM complete
VF complete
{'QG': {'Random': 0.0002936802937205289, 'Bart': 0.06181900089641459}, 'AG':
{'Random': 0.00019573302599379963, 'Bart': 0.06342681680708925}, 'IAG': {'Rando
m': 0.0021857923619408425, 'Bart': 0.09186689122045627}, 'CF': {'Random': 0.0,
'Bart': 0.0}, 'MM': {'Random': 0.0003189032804070459, 'Bart': 0.035383584670417
56}, 'VF': {'Random': 9.536888799123836e-05, 'Bart': 0.005405083401002882}}
```

In [ ]: