

```
!pip install pymongo
```

```
Collecting pymongo
  Downloading pymongo-4.10.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.6.1-py3-none-any.whl.metadata (5.8 kB)
Download pymongo-4.10.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
----- 1.4/1.4 MB 18.1 MB/s eta 0:00:00
Download dnspython-2.6.1-py3-none-any.whl (307 kB)
----- 307.7/307.7 kB 15.8 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.6.1 pymongo-4.10.0
```

```
from pymongo import MongoClient
import os
import pandas as pd
import base64
import numpy as np
from sklearn.metrics import cohen_kappa_score, confusion_matrix
import base64
import matplotlib.pyplot as plt
from collections import defaultdict
import base64
from matplotlib.colors import ListedColormap
plt.rcParams['font.size'] = 14 # You can adjust the size as needed
plt.rcParams['font.family'] = 'DeJavu Serif'
plt.rcParams["font.serif"] = ["Times New Roman"]
plt.rcParams['pdf.fonttype'] = 42
plt.rcParams['ps.fonttype'] = 42

params = {'text.usetex': False, 'mathtext.fontset': 'stixsans'}
plt.rcParams.update(params)
```

```
print('LOAD METADATA!')
```

```
mongodbusuri = 'mongodb+srv://amansinghtha:test123@cluster1.qey3gbo.mongodb.net/?retryWrites=true&w=majority&appName=Cluster1'
client = MongoClient(mongodbusuri)
```

```
benchmark_mapping = {}
model_mapping = {}
evaluator_mapping = {}
```

```
def load_benchmarks():
    db_name = 'metadata'
    coll_name = 'benchmark'
    collection = client[db_name][coll_name]
    benchmark_mapping = {}

    for record in collection.find():
        samples = record['num_samples'] if 'num_samples' in record else 'complete'
        seed = record['seed'] if 'seed' in record else ''
        key = record['name'] + "_" + str(samples) + '_' + str(seed)
        benchmark_mapping[record['_id']] = record['name']
    print('Loaded Benchmarks - ', benchmark_mapping.values())
    return benchmark_mapping
```

```
def load_metadata():
    db_name = 'metadata'
    coll_name = 'model'
    collection = client[db_name][coll_name]
    model_mapping = {}
    for record in collection.find():
        model_mapping[record['_id']] = record['name']
    print('Loaded Metadata - ', model_mapping.keys())
    return model_mapping
```

```
def load_evaluator():
    db_name = 'metadata'
    coll_name = 'evaluator'
    collection = client[db_name][coll_name]
    evaluator_mapping = {}
    for record in collection.find():
        evaluator_mapping[record['_id']] = record['name']
    print('Loaded Evaluator - ', evaluator_mapping.values())
    return evaluator_mapping
```

```
def load_questions():
    db_name = 'datasets'
    coll_name = 'benchmark'
    collection = client[db_name][coll_name]
    question_mapping = {}
    reference_mapping = {}
    for record in collection.find():
        question_mapping[record['_id']] = record['question']
        reference_mapping[record['_id']] = record['references']
    print('Total Question - ', len(question_mapping))
    return question_mapping, reference_mapping
```

```
benchmark_mapping = load_benchmarks()
model_mapping = load_metadata()
evaluator_mapping = load_evaluator()
question_mapping, reference_mapping = load_questions()
```

```
LOAD METADATA!
```

```
Loaded Benchmarks - dict_values(['nq_fewshot4', 'poc-triviaqa-test2', 'triviaQA-llama7b-seed-experiment', 'triviaQA-llama7b-seed-experiment', 'triviaQA-llama13b-seed-experiment', 'triviaQA-llama13b-seed-experiment'])
Loaded Metadata - dict_keys(['6YUZVcdcan1Q2c2NnDtYwAXajNY75QGCh45BBEErRSE=', 'HRHB1fBW5se/dF+j0PsRfGnU0zy3HZwdPKNLlgpboaY=', 'fDzVe5EJU3nCULokR3Wc5G7G+Gr+0filWVeIf'])
Loaded Evaluator - dict_values(['eval-llama7b', 'human-kartik', 'eval-falcon7b', 'contains', 'eval-llama13b', 'exact-uncased', 'eval-gpt-3.5', 'eval-gpt4t', 'eval-gpt4o'])
Total Question - 20262
```

```

print('LOAD EVALUATIONS!')

column_mapping = {
    'exact-match-newline' : 'EM',
    'exact-match' : 'EM',
    'em-uncased' : 'EM',
    'contains' : 'Contains',
    'contains-newline' : 'Contains',
    'human-aman' : 'Human-Aman',
    'human-srinik' : 'Human-Srinik',
    'Human-Srinik' : 'Human-Srinik',
    'multi-human-srinik' : 'Human-Srinik',
    'human-kartik2' : 'Human-Kartik',
    'Human-Kartik' : 'Human-Kartik',
    'multi-human-kartik' : 'Human-Kartik',
    'eval-llama-7b' : 'Llama2-7B',
    'eval-llama7b' : 'Llama2-7B',
    'eval-llama7' : 'Llama2-7B',
    'eval-llama-7b-chat' : 'Llama2-7B',
    'eval-llama-13b' : 'Llama2-13B',
    'eval-llama13b' : 'Llama2-13B',
    'eval-llama-13b-chat' : 'Llama2-13B',
    'eval-llama13' : 'Llama2-13B',
    'eval-llama-70b' : 'Llama2-70B ',
    'eval-llama70b' : 'Llama2-70B',
    'eval-llama70' : 'Llama2-70B',
    'eval-llama-70b-chat' : 'Llama2-70B',
    'eval-judgelm-final2' : 'JudgeLM',
    'eval-judgelm' : 'JudgeLM-7B',
    'eval-gemma2b-it' : 'Gemma-2B',
    'eval-gemma2' : 'Gemma-2B',
    'eval-gemma2-extract' : 'Gemma-2B',
    'eval-phi2-it' : 'Phi2',
    'eval-mistral' : 'Mistral-7B',
    'eval-gpt-4t' : 'GPT-4',
    'eval-llama3-8B-Instruct' : 'Llama3-8B',
    'eval-llama3-8' : 'Llama3-8B',
    'eval-llama3-8-extract' : 'Llama3-8B',
    'eval-llama3-70B-Instruct' : 'Llama3-70B',
    'eval-llama3-70' : 'Llama3-70B',
    'eval-gpt4' : 'GPT-4',
    'eval-gpt4t' : 'GPT-4',
    'eval-gpt4t-human_guidelines' : 'GPT-4 HG',
    'eval-llama3-70B-Instruct-human_guidelines' : 'Llama3-70B HG',
    'eval-gemma2-human_guidelines_non_chat' : 'Gemma-2B HG',
    'eval-llama7b-human_guidelines' : 'Llama2-7B HG',
    'eval-llama3-8-human_guidelines' : 'Llama3-8B HG',
    'eval-mistral-human_guidelines' : 'Mistral-7B HG',
    'eval-llama13b-human_guidelines' : 'Llama2-13B HG',
    'eval-judgelm_human_guidelines' : 'JudgeLM-7B HG',
    'eval-llama3.1-8-human_guidelines' : 'Llama3.1-8B HG',
    'eval-llama3.1-70b-human_guidelines' : 'Llama3.1-70B HG',
    'eval-llama70b-human_guidelines' : 'Llama2-70B HG'
}

model_column = 'model'
non_eval_columns = ['response', 'question', 'references', 'benchmark']

override_model_names = {
    'gpt-4t' : 'GPT-4',
    'llama2-7b-chat' : 'Llama2-7B FT',
    'llama2-7b-base' : 'Llama2-7B Base',
    'mistral-7B' : 'Mistral 7B',
    'llama2-13b-base' : 'Llama2-13B Base',
    'llama2-70b-base' : 'Llama2-70B Base',
    'llama2-13b-chat' : 'Llama2-13B FT',
    'llama2-70b-chat' : 'Llama2-70B FT',
    'mistral-7b-chat' : 'Mistral-7B FT'
}

valid_models = list(override_model_names.values())

human_eval_columns = ['Human-Aman', 'Human-Srinik', 'Human-Kartik']
human_eval_columns_confidence = [human+"_Confidence" for human in human_eval_columns]

print('Preparing Final Dataset!')
benchmark_names = ['triviaQA-400-v1.5.0']

results = defaultdict(list)
benchmark_results = []

def updateForMultiHumanDiscrepancy(data):
    evaluation = data['evaluation'].copy()
    found = False
    for k1 in data['evaluation'].keys():
        if 'other-humans' in data['evaluation'][k1]['info']:
            for other_human in data['evaluation'][k1]['info']['other-humans']:
                evaluation[column_mapping[evaluator_mapping[other_human]]] = data['evaluation'][k1]['info']['other-humans'][other_human]
                #print('Updated ID - ', data['_id'])
                found = True
    data['evaluation'] = evaluation

counter = 0

for i in range(len(benchmark_names)):
    db_name = 'benchmark'
    collection = client[db_name]['TriviaQA']

    benchmarks = []

```

```

for b in benchmark_mapping.keys():
    if benchmark_mapping[b] == benchmark_names[i]:
        query = {'benchmark': b }
        counter = 0
        for record in collection.find(query):
            results[record['model']].append(record)

        data = {
            '_id': record['_id'],
            'model': override_model_names[model_mapping[record['model']]] if model_mapping[record['model']] in override_model_names else model_mapping[record['model']],
            'prompt': record['prompt'],
            'response': record['response'],
            'question': question_mapping[record['question']],
            'references': '\n'.join(reference_mapping[record['question']]),
            'benchmark': benchmark_names[i],
            'evaluation': { column_mapping[evaluator_mapping[key]] if evaluator_mapping[key] in column_mapping else evaluator_mapping[key] : record['evaluation'][key]
        }
        updateForMultiHumanDiscrepancy(data)
        benchmark_results.append(data)
        counter += 1

```

```

df = pd.json_normalize(benchmark_results)
df.set_index(['_id'], drop=True, inplace=True)
df['response'] = df['response'].apply(lambda x: base64.b64decode(x).decode('utf-8'))
df['response'] = df['prompt'].apply(lambda x: base64.b64decode(x).decode('utf-8'))

```

```
print(len(df))
```

LOAD EVALUATIONS!  
Preparing Final Dataset!  
5974

```
columns = []
column_remap = {}
```

```
#Limit only 400 rows
```

```

def limit_400_rows(df):
    models = df[model_column].unique()
    sliced_dfs = []
    for model in models:
        sliced_df = df[df[model_column] == model].head(400)
        sliced_dfs.append(sliced_df)

```

```

sliced_result = pd.concat(sliced_dfs)
return sliced_result

```

```

for col in df.columns:
    for c in human_eval_columns:
        if 'evaluation.'+c+'.info.confident' in col:
            columns.append(col)
            column_remap[col] = c+'_Confidence'
            break
    for ref in column_mapping.keys():
        if 'evaluation.'+column_mapping[ref]+'.result' in col:
            columns.append(col)
            column_remap[col] = column_mapping[ref]
            break

```

```

df = df[[model_column]+non_eval_columns+columns]
df = df[df[model_column].isin(valid_models)]
df = df.rename(columns=column_remap)
print(df.columns)
df.shape

```

Index(['model', 'response', 'question', 'references', 'benchmark', 'EM',  
'Contains', 'Llama2-7B', 'Llama2-13B', 'Llama2-70B', 'Mistral-7B',  
'Human-Srinik', 'Human-Srinik\_Confidence', 'Llama3-8B', 'Llama3-70B',  
'Gemma-2B', 'JudgeLM-7B', 'GPT-4', 'GPT-4 HG', 'Llama3-70B HG',  
'Llama2-7B HG', 'Llama3-8B HG', 'Llama2-13B HG', 'Gemma-2B HG',  
'JudgeLM-7B HG', 'Llama3.1-8B HG', 'Llama3.1-70B HG', 'Mistral-7B HG',  
'Llama2-70B HG', 'Human-Aman', 'Human-Aman\_Confidence', 'Human-Kartik',  
'Human-Kartik\_Confidence'],  
dtype='object')  
(4374, 33)

```

def calculateHumanJudgement(row):
    judgments = row[human_eval_columns]
    num_nan = judgments.isna().sum()

    if num_nan == 2:
        return judgments.dropna().iloc[0]
    else:
        true_count = judgments.sum()
        false_count = len(judgments) - true_count
        if true_count > false_count:
            return True
        elif true_count < false_count:
            return False
        else: # Handle tie
            return judgments.iloc[0] #

```

```

#Testing
# temp = df[df.index == 'm3Dci+PuxgV4araoadprM0VOHzq6d805pivQDqzJK8U='][human_eval_columns]
# print(temp)
# print(temp.apply(calculateHumanJudgement, axis=1))

```

```

df = df[~df[human_eval_columns].isna().all(axis=1)]
df = limit_400_rows(df)

```

```
df['HumanJudgement'] = df.apply(calculateHumanJudgement, axis =1)
df.shape
```

```
(3600, 34)
```

```
df = df.drop(columns=human_eval_columns+human_eval_columns_confidence)
df_non_eval = df.drop(columns=non_eval_columns, axis=1)
```

```
df_non_eval['model'].unique()
```

```
array(['Llama2-70B FT', 'Llama2-13B FT', 'Llama2-7B FT', 'Mistral-7B FT',
       'GPT-4', 'Llama2-70B Base', 'Llama2-7B Base', 'Llama2-13B Base',
       'Mistral 7B'], dtype=object)
```

```
from sklearn.metrics import cohen_kappa_score
import pandas as pd
import numpy as np
from scipy.stats import chi2_contingency
```

```
def calculateHumanAlignment(df_non_eval, refcol, gt, model=None):
    temp = df_non_eval.dropna(subset=[refcol, gt])
    if model:
        temp = temp[temp['model'] == model]
    temp = temp[[refcol, gt]].astype(int)

    ref_values = temp[refcol].to_numpy()
    gt_values = temp[gt].to_numpy()
    #print(len(ref_values), len(gt_values), np.unique(ref_values), np.unique(gt_values))
    kappa_score = cohen_kappa_score(ref_values, gt_values)
    return kappa_score*100
```

```
def generate_random_points(mean, std_dev, n):
    # Generate random points with normal distribution
    return np.random.normal(mean, std_dev, n)
```

```
def calculatePercentageAlignment(df_non_eval, refcol, gt, model=None):
    temp = df_non_eval.dropna(subset=[refcol, gt])
    if model:
        temp = temp[temp['model'] == model]
    temp = temp[[refcol, gt]].astype(int)
    aligned = len(temp[temp[refcol] == temp[gt]])
    total = len(temp)
    return aligned*100/total
```

```
def calculateScore(df_non_eval, refcol, model=None):
    temp = df_non_eval.dropna(subset=[refcol])
    if model:
        temp = temp[temp['model'] == model]
    temp = temp[[refcol]].astype(int)
    return temp.mean().values[0]*100
```

```
#Scott-Pi and #MCC-Pi
```

```
#Can be used for multiple categories
```

```
def scotts_pi(df, col1, col2):
    conf_matrix = pd.crosstab(df[col1], df[col2])
    n = len(df)

    obs_a = np.sum(np.diag(conf_matrix))/n
    # print(obs_a)

    row_sum = conf_matrix.sum(axis=1).tolist()
    col_sum = conf_matrix.sum(axis=0).tolist()

    p_i = (np.array(row_sum) + np.array(col_sum))/(2*n)
    exp_a = np.sum(p_i**2)
    # print(exp_a)
    if exp_a == 1:
        print("100% annotators agreement but both use only 1 category")

    scotts_pi = (obs_a - exp_a)/(1 - exp_a)

    return scotts_pi*100
```

```
#Only valid for 2x2 confusion matrices
```

```
def mcc_phi_r_chi(df, col1, col2):
    conf_matrix = pd.crosstab(df[col1], df[col2])
    n = len(df)

    #Phi and Normalised Chi
    chi2, p_value, dof, expected = chi2_contingency(conf_matrix, correction=False)
    phi = np.sqrt(chi2/n)
    norm_chi = np.sqrt(chi2/(n*(min(conf_matrix.shape)-1)))

    #MCC
    tn, fp, fn, tp = conf_matrix.values.ravel()
    mcc = (tp*tn - fp*fn)/np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))

    #r
    r = np.corrcoef(df[col1], df[col2])[0,1]

    return {
        "MCC": mcc,
        "Phi": phi,
        "r": r,
        "Normalised Chi": norm_chi
    }
```

```
#Only valid for 2x2 confusion matrices
```

```

def calculate_mcc(df, col1, col2):
    conf_matrix = pd.crosstab(df[col1], df[col2], dropna=False).reindex(index=[0, 1], columns=[0, 1], fill_value=0)
    n = len(df)

    #MCC
    tn, fp, fn, tp = conf_matrix.values.ravel()
    mcc = (tp*tn - fp*fn)/np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))

    if np.isnan(mcc): return 0

    return mcc

gt = 'HumanJudgement'

##### Kappa #####

judges = list(set(df_non_eval.columns) - set(['model', gt]))
llm_alignment = {'Human Alignment' : 98.33}
for col in judges:
    #print(col, df_non_eval[col].value_counts(dropna=False))
    llm_alignment[col] = calculateHumanAlignment(df_non_eval, col, gt)
llm_alignment = pd.json_normalize(llm_alignment)
llm_alignment = llm_alignment.sort_values(by=llm_alignment.index[0], axis=1)

llm_alignment_model = pd.DataFrame(index = df_non_eval['model'].unique())

for col in judges:
    values = []
    for model in df_non_eval['model'].unique():
        values.append(calculateHumanAlignment(df_non_eval, col, gt, model=model))
    llm_alignment_model[col] = values

llm_alignment_model['Human Alignment'] = generate_random_points(96.36, 1.67, len(df_non_eval['model'].unique()))

llm_alignment_model = pd.DataFrame(llm_alignment_model)
llm_alignment

##### Percentage Alignment #####
percentage_alignment = {'Human Alignment' : 98.33}
for col in judges:
    values = []
    for model in df_non_eval['model'].unique():
        #print(col, model)
        values.append(calculatePercentageAlignment(df_non_eval, col, gt, model=model))
    percentage_alignment[col] = values

percentage_alignment['Human Alignment'] = generate_random_points(98.33, 0.76, len(df_non_eval['model'].unique()))
percentage_alignment = pd.DataFrame(percentage_alignment)

percentage_alignment.index = llm_alignment_model.index
percentage_alignment

percentage_alignment.mean()

##### Scotts PI #####
scotts_pi_alignment = {}

for col in judges:
    values = []
    for model in df_non_eval['model'].unique():
        df_uniq_model = df_non_eval[df_non_eval['model'] == model]
        values.append(scotts_pi(df_uniq_model, col, gt))
    scotts_pi_alignment[col] = values

scotts_pi_alignment = pd.DataFrame(scotts_pi_alignment)
scotts_pi_alignment.index = df_non_eval['model'].unique()
scotts_pi_alignment

##### MCC #####

mcc_alignment = {}

for col in judges:
    values = []
    for model in df_non_eval['model'].unique():
        df_uniq_model = df_non_eval[df_non_eval['model'] == model]
        values.append(calculate_mcc(df_uniq_model, col, gt))
    mcc_alignment[col] = values

mcc_alignment = pd.DataFrame(mcc_alignment)
mcc_alignment.index = df_non_eval['model'].unique()
mcc_alignment

# print(scotts_pi(df, 'human1', 'human2'))

# df_non_eval[judges]

##### Score Matrix #####

score_matrix = {}
for col in judges+['HumanJudgement']:
    values = []
    for model in df_non_eval['model'].unique():
        values.append(calculateScore(df_non_eval, col, model=model))
    score_matrix[col] = values

score_matrix = pd.DataFrame(score_matrix)
score_matrix.index = llm_alignment_model.index
score_matrix['Human Alignment'] = score_matrix['HumanJudgement']
score_matrix.T

```

```
<ipython-input-10-a7b436b61dc8>:92: RuntimeWarning: invalid value encountered in scalar divide
mcc = (tp*tn - fp*fn)/np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
<ipython-input-10-a7b436b61dc8>:92: RuntimeWarning: invalid value encountered in scalar divide
mcc = (tp*tn - fp*fn)/np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
```

	Llama2-70B FT	Llama2-13B FT	Llama2-7B FT	Mistral-7B FT	GPT-4	Llama2-70B Base	Llama2-7B Base	Llama2-13B Base	Mistral 7B	
Llama2-70B HG	81.25	74.75	67.50	72.50	96.75	90.25	71.25	80.500000	80.00	
Llama3.1-8B HG	75.50	70.50	60.25	59.00	89.00	83.50	65.25	75.000000	73.75	
Llama2-13B HG	77.50	62.75	63.00	67.50	93.50	86.50	68.25	75.500000	74.50	
Llama3-8B HG	85.25	82.75	73.00	76.00	97.25	91.50	76.00	83.250000	81.75	
Mistral-7B	77.50	68.75	64.50	68.50	92.75	85.25	67.75	75.000000	74.50	
GPT-4 HG	73.00	59.00	54.25	56.50	90.00	82.50	60.50	71.500000	69.75	
Llama3-8B	78.50	75.00	52.75	66.00	76.25	85.25	68.75	76.000000	62.50	
Llama2-13B	39.50	27.75	45.75	39.00	73.50	66.50	53.00	61.500000	57.50	
Gemma-2B HG	68.50	41.00	58.00	55.75	80.50	91.25	79.75	87.000000	84.00	
Llama3.1-70B HG	74.00	64.75	55.50	60.50	92.25	85.00	62.00	74.250000	72.25	
Gemma-2B	75.25	44.25	100.00	79.50	99.75	84.00	67.00	74.000000	100.00	
Llama2-70B	82.25	80.00	64.00	74.25	95.25	86.50	68.25	76.296296	77.00	
Mistral-7B HG	82.50	74.75	69.00	72.00	96.25	90.50	72.50	80.750000	80.25	
Contains	59.50	46.25	38.75	44.00	70.00	68.00	50.75	60.000000	57.25	
GPT-4	72.00	57.50	56.50	52.75	92.25	85.00	63.25	75.000000	72.50	
Llama3-70B HG	75.75	64.00	57.00	62.50	92.75	86.50	64.25	75.500000	73.50	
Llama2-7B	58.00	77.25	58.50	80.50	88.00	80.25	62.50	71.250000	68.25	
JudgeLM-7B HG	82.75	48.00	63.25	71.00	94.50	86.25	69.50	77.750000	77.25	
Llama2-7B HG	90.75	70.75	80.00	83.25	97.75	92.00	80.50	85.250000	84.00	
EM	36.25	0.25	24.00	20.25	58.25	63.75	46.75	56.000000	59.50	
Llama3-70B	78.25	68.50	61.50	65.00	94.50	87.00	66.00	76.500000	73.50	
JudgeLM-7B	60.00	32.50	41.25	45.50	68.50	64.00	52.50	60.500000	57.25	
HumanJudgement	72.25	56.50	56.00	60.75	91.50	83.75	62.25	72.750000	71.75	
Human Alignment	72.25	56.50	56.00	60.75	91.50	83.75	62.25	72.750000	71.75	

score\_matrix.T.to\_csv('score\_matrix.csv')

scotts\_pi\_alignment.T.to\_csv('scotts\_pi\_alignment.csv')

scotts\_pi\_alignment.T

	Llama2-70B FT	Llama2-13B FT	Llama2-7B FT	Mistral-7B FT	GPT-4	Llama2-70B Base	Llama2-7B Base	Llama2-13B Base	Mistral 7B	
Llama2-70B HG	73.380967	57.333333	74.597126	70.206098	43.488458	71.264368	77.471769	76.969581	77.464981	
Llama3.1-8B HG	82.512812	63.326502	83.052480	74.505557	63.065559	80.830452	82.691007	81.217465	86.128188	
Llama2-13B HG	71.428334	70.403256	78.213508	67.941534	56.756757	75.320525	82.358940	85.010305	82.826476	
Llama3-8B HG	59.663866	37.939136	60.694399	60.113013	45.842531	64.264615	66.615663	67.948718	71.979966	
Mistral-7B	71.428334	66.354916	78.078760	75.394878	36.249488	61.824776	79.120879	78.626770	76.465911	
GPT-4 HG	90.568929	87.704603	92.420366	81.963295	76.174522	83.068783	92.090641	91.917371	92.751654	
Llama3-8B	64.307060	51.153174	69.264683	69.303472	17.743967	65.642298	76.767342	73.109244	60.915097	
Llama2-13B	19.387037	35.910172	75.492495	38.499616	25.541126	43.143337	72.869036	65.446680	61.178585	
Gemma-2B HG	33.448749	19.949969	26.560588	27.012927	16.943522	15.428571	36.862555	24.571584	33.980829	
Llama3.1-70B HG	91.731266	77.483227	92.906184	82.719686	78.231293	89.570370	93.093877	92.298806	93.799603	
Gemma-2B	63.841808	7.494797	541.640379	42.123673	1.400560	60.258321	75.394878	72.486823	582.667283	
Llama2-70B	68.704435	44.622307	79.166667	68.091168	53.524791	77.294883	84.564073	48.794243	84.914942	
Mistral-7B HG	66.440257	57.333333	70.133333	68.075946	45.650697	67.683972	74.409827	74.781969	73.958333	
Contains	70.529155	77.482971	64.401883	64.419722	29.234852	56.978600	76.604618	70.316230	67.245332	
GPT-4	90.673889	87.760098	92.888889	77.591607	64.835165	72.503704	91.443621	87.694201	89.430408	
Llama3-70B HG	88.305613	82.254234	91.862476	81.499957	84.493119	85.192315	92.471297	91.527564	94.341358	
Llama2-7B	33.406256	46.393934	74.463086	43.965355	56.518785	67.818428	83.488572	78.918651	77.380952	
JudgeLM-7B HG	59.856631	55.910719	70.403256	63.856510	34.715822	72.549020	76.089691	77.180442	77.628635	
Llama2-7B HG	35.334107	41.128417	41.406250	42.956349	28.727228	58.938863	54.114057	59.312839	61.549274	
EM	27.476014	-38.385958	33.333333	15.966387	11.627173	48.345440	68.746849	63.481519	72.848485	
Llama3-70B	79.865096	71.200000	85.557705	76.973178	61.597542	76.974384	85.329177	83.497124	89.311453	
JudgeLM-7B	58.147029	43.314101	62.471619	61.349020	23.437500	44.947741	71.888397	62.336011	59.602577	

