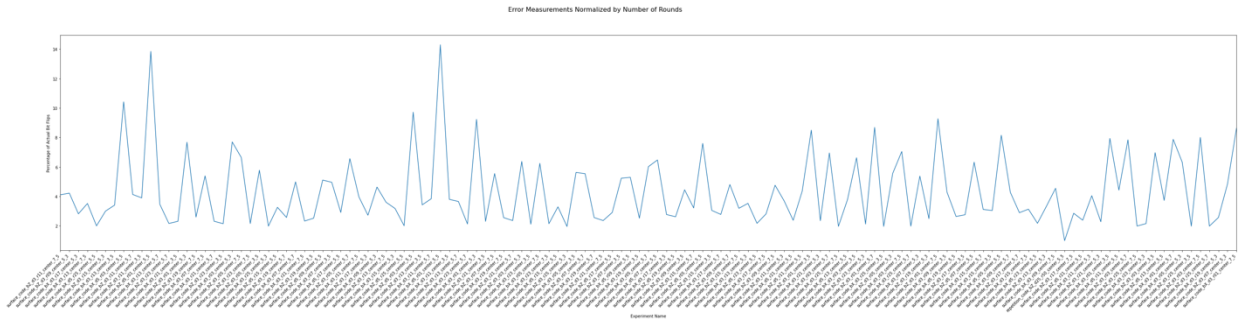
Google QEC Data Dive

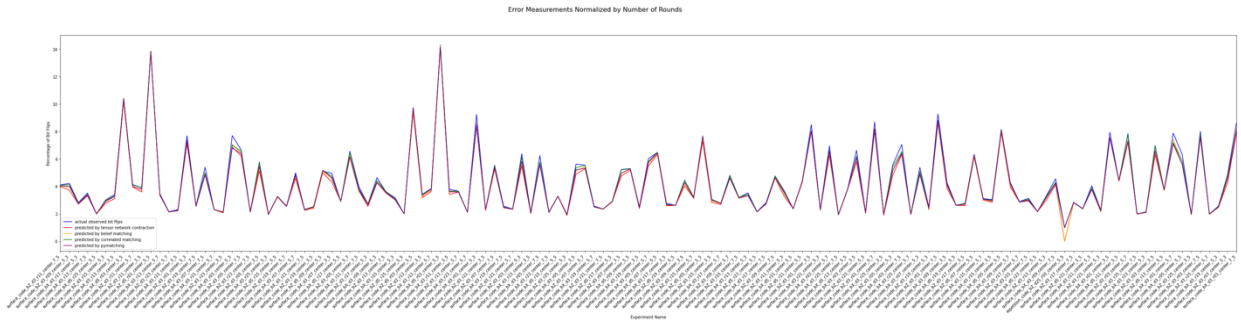
```
import os
import yaml
import matplotlib.pyplot as plt


# Providing the path
path = '/Users/selalipotakey/downloads/google_qec3v5_experiment_data'
flips_path = ""
yaml_path = ""
all_flips = {}

experiments = os.listdir(path)

for folder in experiments:
    flips_path = os.path.join(path, folder, 'obs_flips_actual.01')
    yaml_path = os.path.join(path, folder, 'properties.yml')
    count = 0
    data = 0
    if os.path.isdir(os.path.join(path, folder)):
        with open(flips_path, 'r') as f:
            while (byte := f.read(1)):
                if byte == '1':
                    count += 1
        with open(yaml_path, 'r') as f:
            info = yaml.safe_load(f)
        data = info['shots']
        # update all_flips dictionary
        all_flips[folder] = [count, data]

# turning the dictionary into a graph
x_axis = []
y_axis = []
for entry in all_flips:
    x_axis.append(entry)
    y_axis.append(((all_flips[entry][0]/int(entry.split('_')[4][1:]))/all_flips[entry][1]) * 100)
# dividing by the substring corresponding with number of rounds lets us normalize error rate by
the
# number of rounds
plt.figure().set_figheight(10)
plt.plot(x_axis, y_axis)
plt.margins(x=0)
plt.xticks(rotation=45, ha="right")
plt.subplots_adjust(right=7)
```

```
plt.ylabel("Percentage of Actual Bit Flips")
plt.xlabel("Experiment Name")
plt.suptitle("Error Measurements Normalized by Number of Rounds", x=3.25, fontsize=16)
plt.savefig("bitflips_normalized.png", facecolor='white', bbox_inches="tight",
        pad_inches=0.3, transparent=True)
plt.show()
```




```
all_flips1 = {}
for folder in experiments:
   all_paths = [os.path.join(path, folder, 'obs_flips_actual.01'),
            os.path.join(path, folder, 'obs_flips_predicted_by_tensor_network_contraction.01'),
            os.path.join(path, folder, 'obs_flips_predicted_by_belief_matching.01'),
            os.path.join(path, folder, 'obs_flips_predicted_by_correlated_matching.01'),
            os.path.join(path, folder, 'obs_flips_predicted_by_pymatching.01'),
            os.path.join(path, folder, 'properties.yml')]
   count = 0
   if os.path.isdir(os.path.join(path, folder)):
      all_flips1[folder] = []
      for i in range(len(all_paths)):
         count = 0
         if i != (len(all_paths)-1):
            if os.path.isfile(all_paths[i]) != True:
               count = 0
            else:
               with open(all_paths[i], 'r') as f:
                  while (byte := f.read(1)):
                     if byte == '1':
                        count += 1
               count = count/int(folder.split('_')[4][1:])
         else:
            with open(all_paths[i], 'r') as f:
               info = yaml.safe_load(f)
            count = info['shots']
         all_flips1[folder].append(count)
```

```
y_tensor = [] #1
y_belief = [] #2
y_correlated = [] #3
y_pymatching = [] #4
for entry in all_flips1:
    y_tensor.append((all_flips1[entry][1]/all_flips[entry][-1])*100)
    y_belief.append((all_flips1[entry][2]/all_flips[entry][-1])*100)
    y_correlated.append((all_flips1[entry][3]/all_flips[entry][-1])*100)
    y_pymatching.append((all_flips1[entry][4]/all_flips[entry][-1])*100)

fig = plt.figure()
fig.set_figheight(10)
# plotting
ax1 = fig.add_subplot(111)
ax1.plot(x_axis, y_axis, c='b', label="actual observed bit flips")
ax1.plot(x_axis, y_tensor, c='r', label="predicted by tensor network contraction")
ax1.plot(x_axis, y_belief, c='orange', label="predicted by belief matching")
ax1.plot(x_axis, y_correlated, c='green', label="predicted by correlated matching")
ax1.plot(x_axis, y_pymatching, c='purple', label="predicted by pymatching")
plt.margins(x=0)
plt.xticks(rotation=45, ha="right")
plt.subplots_adjust(right=7)
plt.ylabel("Percentage of Bit Flips")
plt.xlabel("Experiment Name")
plt.legend(loc='lower left')
plt.suptitle("Error Measurements Normalized by Number of Rounds", x=3.25, fontsize=16)
plt.savefig("all_bitflips_normalized.png", facecolor='white', bbox_inches="tight",
        pad_inches=0.3, transparent=True)
plt.show()
```




```
# for each key-value pair, check that key includes d3 or d5, two different plots
dist_3 = {}
dist_5 = {}
rounds = ['r01','r03','r05','r07', 'r09', 'r11', 'r13', 'r15', 'r17', 'r19', 'r21', 'r23', 'r25']
for each in rounds:
```

```
    dist_3[each] = []
    dist_5[each] = []
temp = []

for entry in all_flips:
    if "d3" in entry:
        temp = entry.split('_')
        dist_3[temp[4]].append([entry,
((all_flips[entry][0]/int(temp[4][1:]))/all_flips[entry][1])*100])
    if "d5" in entry:
        temp = entry.split('_')
        dist_5[temp[4]].append([entry,
((all_flips[entry][0]/int(temp[4][1:]))/all_flips[entry][1])*100])

d3_points = [[],[]]
d5_points = [[],[]]

for entry in dist_3:
    for each in dist_3[entry]:
        d3_points[0].append(each[0])
        d3_points[1].append(each[1])
for entry in dist_5:
    for each in dist_5[entry]:
        d5_points[0].append(each[0])
        d5_points[1].append(each[1])

figure, axis = plt.subplots(1,2, gridspec_kw={'width_ratios': [4, 1]})
figure.set_figheight(10)
axis[0].plot(d3_points[0],d3_points[1])
axis[0].set_title("Distance 3 Codes")
axis[0].set_xticklabels(d3_points[0], rotation=45, ha='right')
axis[0].margins(x=0)
axis[1].plot(d5_points[0],d5_points[1])
axis[1].set_title("Distance 5 Codes")
axis[1].set_xticklabels(d5_points[0], rotation=45, ha='right')
axis[1].margins(x=0)
plt.subplots_adjust(right=5)
plt.suptitle('Error Measurements Normalized by Rounds', x=2.35,  fontsize=16)
figure.text(2.5, -0.1, "Experiment Name", ha='center', va='center')
figure.text(0, 0.5, "Percent Error Rate", ha='center', va='center', rotation='vertical')
plt.savefig("rounds_normalized.png", facecolor='white', bbox_inches="tight",
        pad_inches=0.3, transparent=True)
plt.show()
```

Error Measurements Normalized by Rounds