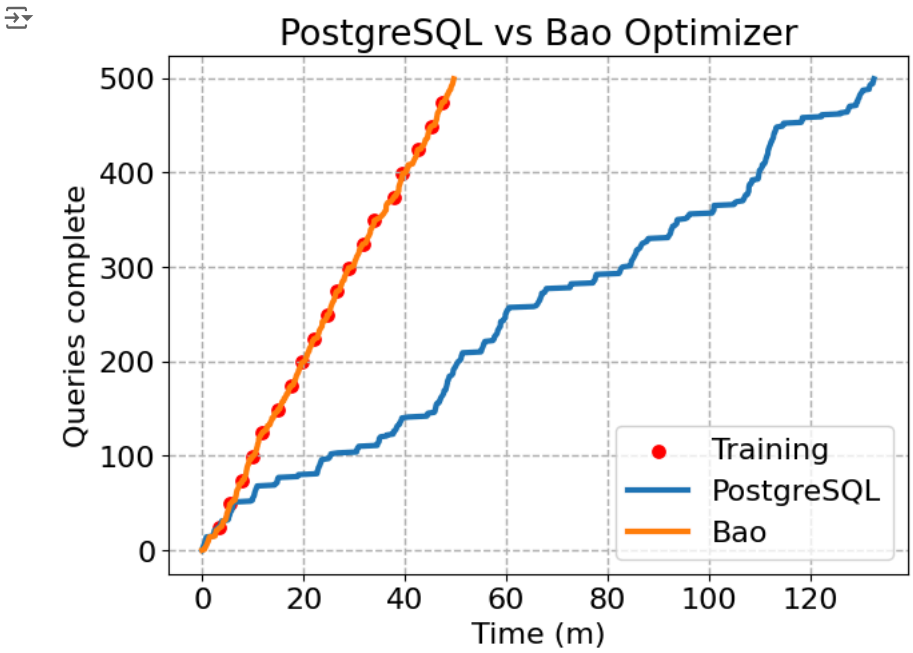
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
import string
from collections import defaultdict
```

```
plt.rcParams["font.size"] = 16
SHOW_RG = False
src_fp = f'/Users/saharshbarve/Work/uiuc/study/Semester_3/cs511/project-bao/breakingBao/exp3-schema-drop-idx/20perct_drop_inde
bao_run_path = f'{src_fp}/bao_run.txt'
pg_run_path = f'{src_fp}/pg_run.txt'
```

```
with open(pg_run_path) as f:
    data = f.read().split("\n")[2:]
data = [x.split(" ") for x in data if len(x) > 1 and (x[0] in string.digits or x[0] == "x")]

data = [(x[0], x[1], float(x[2]), x[3], float(x[4])) for x in data]
pg_data = data
pg_times = np.array([x[2] for x in pg_data])
pg_times -= np.min(pg_times)
pg_times /= 60


def read_bao_data(fp):
    with open(fp) as f:
        data = f.read().split("\n")[2:]

    training_times = []
    for idx in range(len(data)):
        if data[idx].strip().startswith("Initial input channels"):
            prev_line = data[idx-1].split(" ")
            if prev_line[0] == "Retry":
                continue
            training_times.append(float(prev_line[2]))


    training_times = np.array(training_times)

    data = [x.split(" ") for x in data if len(x) > 1 and (x[0] in string.digits or x[0] == "x")]
    data = [(x[0], x[1], float(x[2]), x[3], float(x[4])) for x in data]
    bao_data = data

    bao_times = np.array([x[2] for x in bao_data])
    training_times -= np.min(bao_times)
    bao_times -= np.min(bao_times)

    bao_times /= 60
    training_times /= 60
    return bao_data, bao_times, training_times

bao_data, bao_times, training_times = read_bao_data(bao_run_path)
if SHOW_RG:
    bao_rb_data, bao_rb_times, training_rb_times = read_bao_data("bao_with_regblock.txt")
```

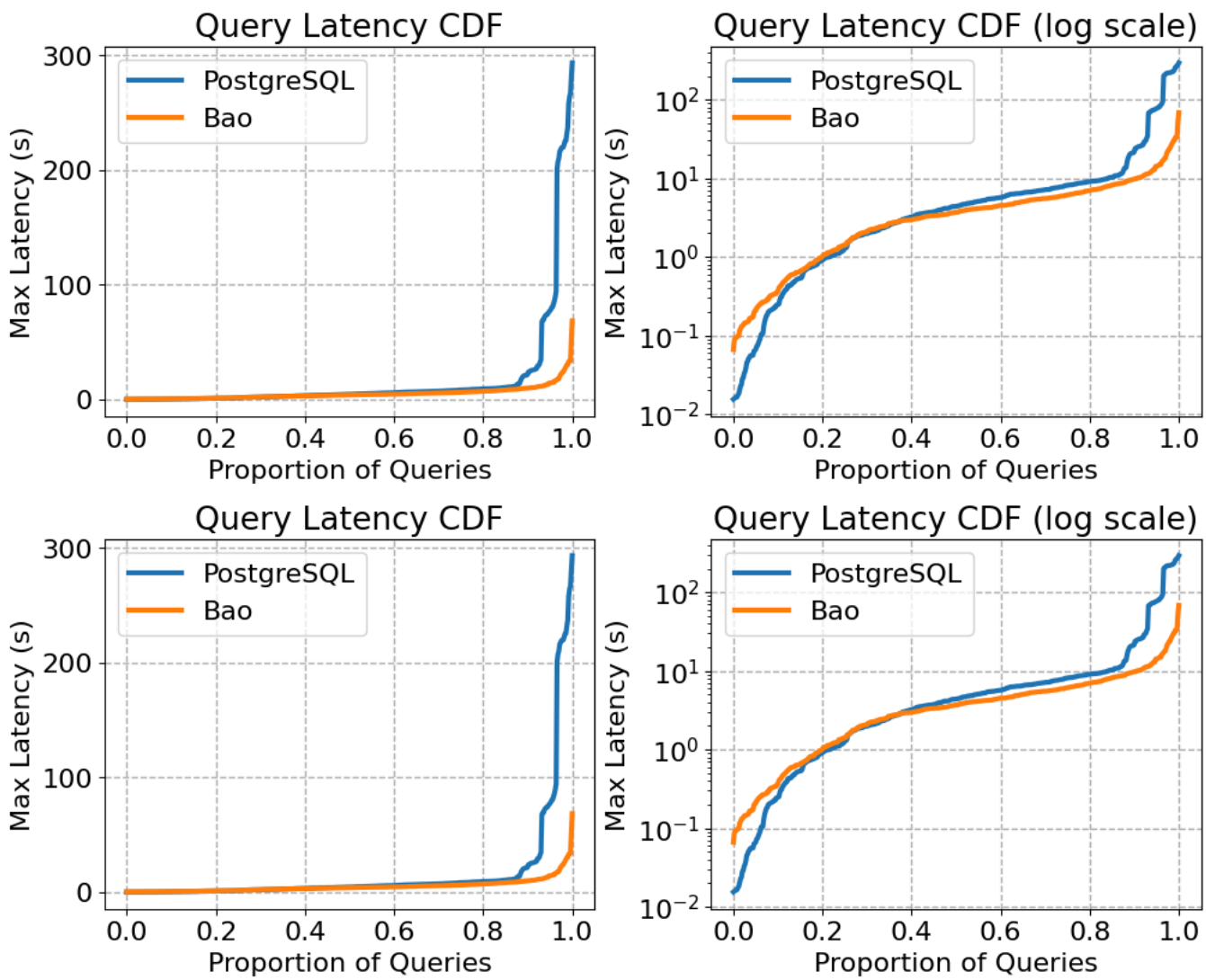
```
queries_complete = np.arange(0, len(pg_times))

fig, ax = plt.subplots(1, 1, constrained_layout=True)


train_y = []
train_rb_y = []
for tt in training_times:
    idx = np.searchsorted(bao_times, tt)
    train_y.append(idx)

if SHOW_RG:
    for tt in training_rb_times:
        idx = np.searchsorted(bao_rb_times, tt)
        train_rb_y.append(idx)

plt.scatter(training_times, train_y, s=45, color="red", label="Training")

```

```
ax.plot(pg_times, queries_complete, label="PostgreSQL", lw=3)
ax.plot(bao_times, queries_complete, label="Bao", lw=3)

if SHOW_RG:
    plt.scatter(training_rb_times, train_rb_y, s=45, color="red")
    ax.plot(bao_rb_times, queries_complete, label="Bao (w/ exploration)", lw=3)

ax.set_xlabel("Time (m)")
ax.set_ylabel("Queries complete")
ax.set_title("PostgreSQL vs Bao Optimizer")

ax.grid(linestyle="--", linewidth=1)
ax.legend()
fig.savefig("queries_vs_time.svg")
```




```
all_pg_times = sorted([x[4] for x in pg_data])
all_bao_times = sorted([x[4] for x in bao_data])

if SHOW_RG:
    all_bao_rb_times = sorted([x[4] for x in bao_rb_data])


fig, axes = plt.subplots(1, 2, figsize=(10, 4), constrained_layout=True)

ax = axes[0]
ax.plot(np.linspace(0, 1, len(all_pg_times)), all_pg_times, lw=3, label="PostgreSQL")
ax.plot(np.linspace(0, 1, len(all_pg_times)), all_bao_times, lw=3, label="Bao")

if SHOW_RG:
    ax.plot(np.linspace(0, 1, len(all_pg_times)), all_bao_rb_times, lw=3, label="Bao (w/ exploration)")

ax.grid(linestyle="--", linewidth=1)
ax.set_xlabel("Proportion of Queries")
ax.set_ylabel("Max Latency (s)")
ax.set_title("Query Latency CDF")
ax.legend()
#ax.set_yscale("log")


ax = axes[1]
ax.plot(np.linspace(0, 1, len(all_pg_times)), all_pg_times, lw=3, label="PostgreSQL")
ax.plot(np.linspace(0, 1, len(all_pg_times)), all_bao_times, lw=3, label="Bao")

if SHOW_RG:
    ax.plot(np.linspace(0, 1, len(all_pg_times)), all_bao_rb_times, lw=3, label="Bao (w/ exploration)")

ax.grid(linestyle="--", linewidth=1)
ax.set_xlabel("Proportion of Queries")
ax.set_ylabel("Max Latency (s)")
ax.set_title("Query Latency CDF (log scale)")
```

```
ax.legend()
ax.set_yscale("log")
fig.savefig("cdf.svg")
fig
```




```
# get the last PG time for each query
pg_query_time = {}
for itm in pg_data:
    pg_query_time[itm[3]] = itm[4]

# get each Bao time
bao_query_times = defaultdict(list)
for itm in bao_data[50:]:
    bao_query_times[itm[3]].append(itm[4])

if SHOW_RG:
    # get each Bao time
    bao_rb_query_times = defaultdict(list)
    for itm in bao_rb_data[50:]:
        bao_rb_query_times[itm[3]].append(itm[4])

max_repeats = max(len(x) for x in bao_query_times.values())

def extract_q_number(x):
    return int(x[x.find("/q")+2:x.find("_", x.find("/q"))])

q_order = sorted(bao_query_times.keys(), key=extract_q_number)

grid = [bao_query_times[x] for x in q_order]

if SHOW_RG:
    grid_rb = [bao_rb_query_times[x] for x in q_order]
```

```
28
29
30 reg_data = []
31 for idx, q in enumerate(q_order):
32     if SHOW_RG:
33         reg_data.append({"Q": f"q{extract_q_number(q)}",
34                          "PG": pg_query_time[q],
35                          "Bao worst": max(grid[idx]),
36                          "Bao best": min(grid[idx]),
37                          "Bao + E worst": max(grid_rb[idx]),
38                          "Bao + E best": min(grid_rb[idx])})
39     else:
40         reg_data.append({"Q": f"q{extract_q_number(q)}",
41                  "PG": pg_query_time[q],
42                  "Bao worst": max(grid[idx]),
43                  "Bao best": min(grid[idx])})
44
45
46
47 def color_regression(col):
48     def c_for_diff(diff):
49         if diff < 2 and diff > -2:
50             return "background-color: white"
51         elif diff > 0.5:
52             return "background-color: #f27281"
53         else:
54             return "background-color: #9ee3ad"
55
56     to_r = [""]
57
58     if SHOW_RG:
59         pg, bao_worst, bao_best, bao_rg_worst, bao_rg_best = col
60     else:
61         pg, bao_worst, bao_best = col
62
63
64     to_r.append(c_for_diff(bao_worst - pg))
65     to_r.append(c_for_diff(bao_best - pg))
66
67     if SHOW_RG:
68         to_r.append(c_for_diff(bao_rg_worst - pg))
69         to_r.append(c_for_diff(bao_rg_best - pg))
70
71     return to_r
72
73 reg_data = pd.DataFrame(reg_data).set_index("Q")
74 reg_data.style.apply(color_regression, axis=1)
```

Q	PG	Bao worst	Bao best
q1	219.772419	5.961556	2.324733
q2	73.944566	23.619783	6.217584
q3	2.167162	4.016232	1.214790
q4	9.145486	5.407017	1.904017
q5	5.465291	8.656160	2.045163
q6	13.446092	7.231444	3.661339
q7	17.893606	33.531573	5.071128
q8	10.944437	14.993714	3.429528
q9	4.663881	10.444022	3.312074
q10	3.670723	7.339001	1.626823
q11	8.657063	8.197291	2.917647
q12	9.694067	6.066285	1.740404
q13	5.013691	4.587697	2.030013
q14	3.928585	7.543591	3.697845
q15	3.721831	5.624449	1.916434
q16	6.321914	4.457160	1.517086
q17	1.965472	6.168261	2.976739
q18	1.750170	8.795283	2.755019
q19	7.168235	18.009343	6.167539
q20	0.074397	0.430175	0.196944
q21	0.517834	0.912397	0.467987
q22	26.077392	52.297187	3.496831
q23	4.461288	5.957746	3.887469
q24	0.017517	0.170877	0.132628
q25	1.163748	2.443217	0.899729
q26	3.920475	7.213808	3.507530
q27	0.391639	0.826040	0.280743
q28	0.527900	1.812794	0.794799
q29	0.023764	0.143189	0.086723
q30	3.801883	5.347154	2.762094
q31	0.531534	1.618418	0.551017
q32	0.179611	0.360545	0.251874
q33	0.475122	1.351058	0.636105
q34	1.613504	2.684404	1.378078
q35	2.404320	3.691195	2.281975