Hide code

```
battles
```

```
def compute_mle_elo(df, SCALE=400, BASE=10, INIT_RATING=1000):
    from sklearn.linear_model import LogisticRegression
    models = pd.concat([df["model_a"], df["model_b"]]).unique()
    models = pd.Series(np.arange(len(models)), index=models)

    # duplicate battles
    df = pd.concat([df, df], ignore_index=True)
    p = len(models.index)
    n = df.shape[0]

    X = np.zeros([n, p])
    X[np.arange(n), models[df["model_a"]]] = +math.log(BASE)
    X[np.arange(n), models[df["model_b"]]] = -math.log(BASE)

    # one A win => two A win
    Y = np.zeros(n)
    Y[df["winner"] == "model_a"] = 1.0

    # one tie => one A win + one B win
    # find tie + tie (both bad) index
    tie_idx = (df["winner"] == "tie") | (df["winner"] == "tie (bothbad)")
    tie_idx[len(tie_idx)//2:] = False
    Y[tie_idx] = 1.0

    lr = LogisticRegression(fit_intercept=False, penalty=None, tol=1e-8)
    lr.fit(X,Y)

    elo_scores = SCALE * lr.coef_[0] + INIT_RATING

    # set anchor as mixtral = 1114
    if "mixtral-8x7b-instruct-v0.1" in models.index:
        elo_scores += 1114 - elo_scores[models["mixtral-8x7b-instruct-v0.1"]]
    return pd.Series(elo_scores, index = models.index).sort_values(ascending=False)
```

```
def get_bootstrap_result(battles, func_compute_elo, num_round):
    rows = []
    for i in tqdm(range(num_round), desc="bootstrap"):
        rows.append(func_compute_elo(battles.sample(frac=1.0, replace=True)))
    df = pd.DataFrame(rows)
    return df[df.median().sort_values(ascending=False).index]
```

```
def compute_online_elo(battles, K=4, SCALE=400, BASE=10, INIT_RATING=1000):
```

```
    rating = defaultdict(lambda: INIT_RATING)

    for rd, model_a, model_b, winner in battles[['model_a', 'model_b', 'winner']].itertuples():
        ra = rating[model_a]
        rb = rating[model_b]
        ea = 1 / (1 + BASE ** ((rb - ra) / SCALE))
        eb = 1 / (1 + BASE ** ((ra - rb) / SCALE))
        if winner == "model_a":
            sa = 1
        elif winner == "model_b":
            sa = 0
        elif winner == "tie" or winner == "tie (bothbad)":
            sa = 0.5
        else:
            raise Exception(f"unexpected vote {winner}")
        rating[model_a] += K * (sa - ea)
        rating[model_b] += K * (1 - sa - eb)

    # calibrate llama-13b to 800
    delta = (800-rating["llama-13b"])
    for model in battles["model_a"].unique():
        rating[model] += delta

    return rating
```

	Model	Elo rating
1	### Model A: claude-3-opus-20240229	1256.17
2	### Model B: claude-3-opus-20240229	1226.26
3	### Model B: claude-3-sonnet-20240229	1181.77
4	### Model B: gpt-4-turbo-2024-04-09	1176.08
5	claude-3-opus-20240229	1161.58
...	...	...
224	### Model B: gemma-2b-it	829.69
225	### Model B: gemma-7b-it	822.82
226	### Model A: gemma-2b-it	819.47
227	dolly-v2-12b	811.64
228	llama-13b	800.00

228 rows × 2 columns

```
def preety_print_model_ratings(ratings):
    df = pd.DataFrame([
        [n, ratings[n]] for n in ratings.keys()
    ], columns=["Model", "Elo rating"]).sort_values("Elo rating", ascending=False).reset_index(drop=True)
    # df["Elo rating"] = (df["Elo rating"] + 0.5).astype(int)
    df.index = df.index + 1
    return df

```

```
online_elo_ratings = compute_online_elo(battles)
```

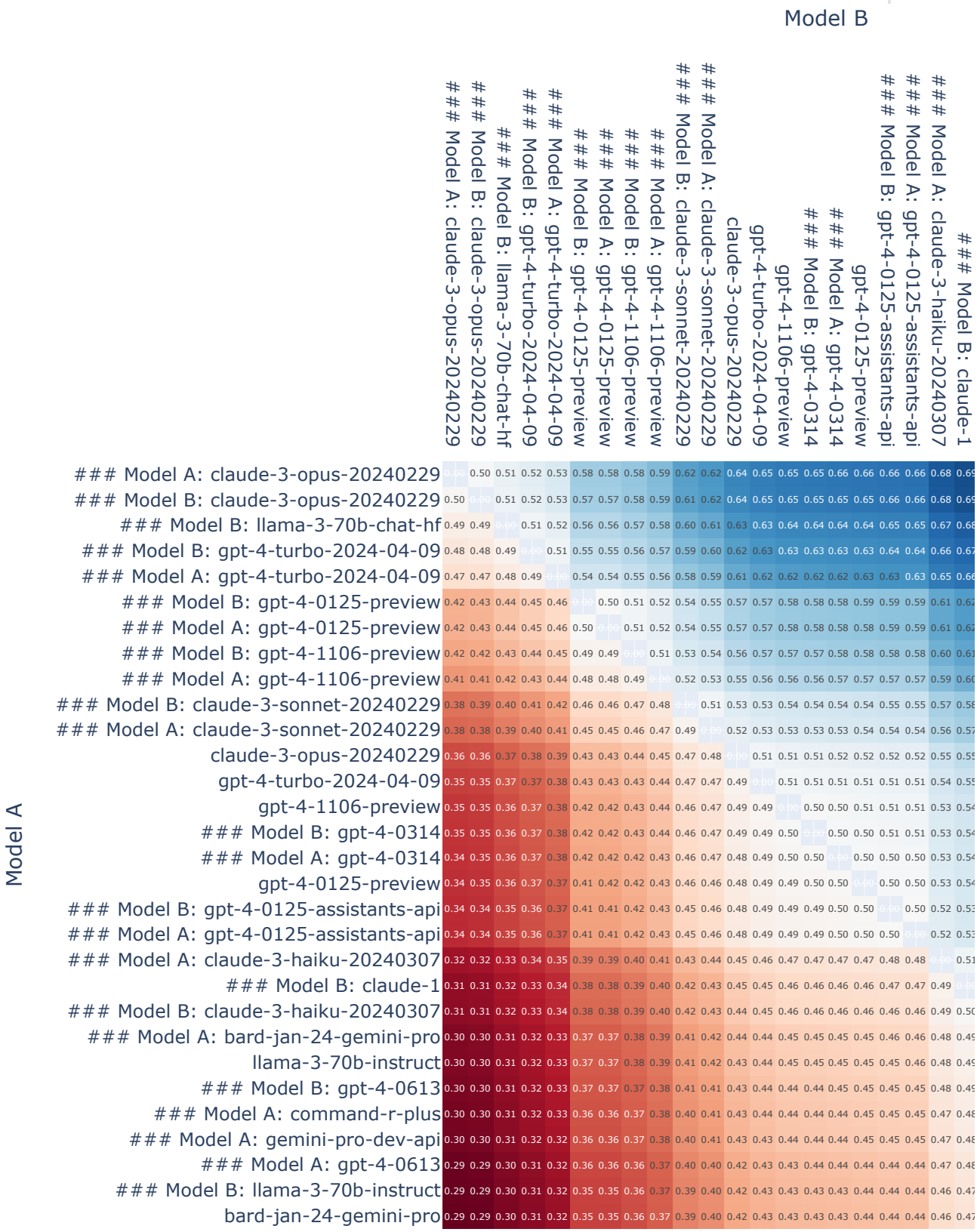
	Model	Elo rating
1	### Model A: claude-3-opus-20240229	1363.86
2	### Model B: claude-3-opus-20240229	1360.53
3	### Model B: gpt-4-turbo-2024-04-09	1340.93
4	### Model A: gpt-4-turbo-2024-04-09	1339.17
5	### Model A: gpt-4-0125-assistants-api	1322.65
...	...	...
224	### Model B: gemma-1.1-2b-it	868.82
225	fastchat-t5-3b	866.37
226	stablelm-tuned-alpha-7b	831.28
227	dolly-v2-12b	809.97
228	llama-13b	792.67

228 rows × 2 columns

```
elo_mle_ratings = compute_mle_elo(battles)
preety_print_model_ratings(elo_mle_ratings)
```

```
BOOTSTRAP_ROUNDS = 100

np.random.seed(42)
bootstrap_elo_lu = get_bootstrap_result(battles, compute_mle_elo, BOOTSTRAP_ROUNDS)
```

```
def predict_win_rate(elo_ratings, SCALE=400, BASE=10, INIT_RATING=1000):
    names = sorted(list(elo_ratings.keys()))
    wins = defaultdict(lambda: defaultdict(lambda: 0))
    for a in names:
        for b in names:
            ea = 1 / (1 + BASE ** ((elo_ratings[b] - elo_ratings[a]) / SCALE))
            wins[a][b] = ea
            wins[b][a] = 1 - ea

    data = {
        a: [wins[a][b] if a != b else np.NAN for b in names]
        for a in names
    }

    df = pd.DataFrame(data, index=names)
    df.index.name = "model_a"
    df.columns.name = "model_b"

```




```
win_rate = predict_win_rate(dict(bootstrap_elo_lu.quantile(0.5)))
ordered_models =
win_rate.mean(axis=1).sort_values(ascending=False).index
ordered_models = ordered_models[:30]
```

```
fig = px.imshow(win_rate.loc[ordered_models, ordered_models],
                color_continuous_scale='RdBu', text_auto=".2f",
                title="Predicted Win Rate Using Elo Ratings for Model A in an A vs. B Battle")
fig.update_layout(xaxis_title="Model B",
                  yaxis_title="Model A",
                  xaxis_side="top", height=900, width=900,
                  title_y=0.07, title_x=0.5)
fig.update_traces(hovertemplate=
                  "Model A: %{y}<br>Model B: %{x}<br>Win Rate: %{z} <extra></extra>")
fig
```

```
from collections import defaultdict
import json, math, gdown
import numpy as np
import pandas as pd
import plotly.express as px
import sklearn
from tqdm import tqdm
import requests
pd.options.display.float_format = '{:.2f}'.format
```

```
url = "https://storage.googleapis.com/arena_external_data/public/clean_battle_20240419.json"
response = requests.get(url)

with open('local_file_name.json', 'wb') as file:
    file.write(response.content)

# load the JSON data from the local file
with open('local_file_name.json', 'r') as file:
    battles = pd.read_json(file).sort_values(ascending=True, by=["tstamp"])
```

```
import marimo as mo
```