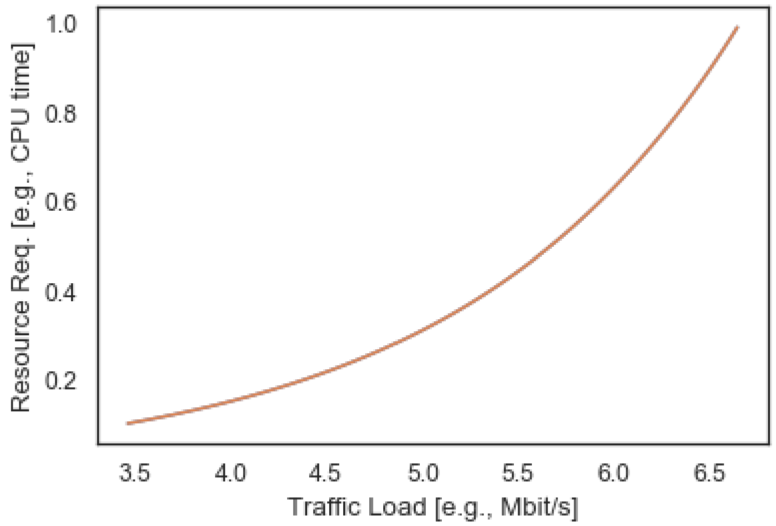
Script-1:

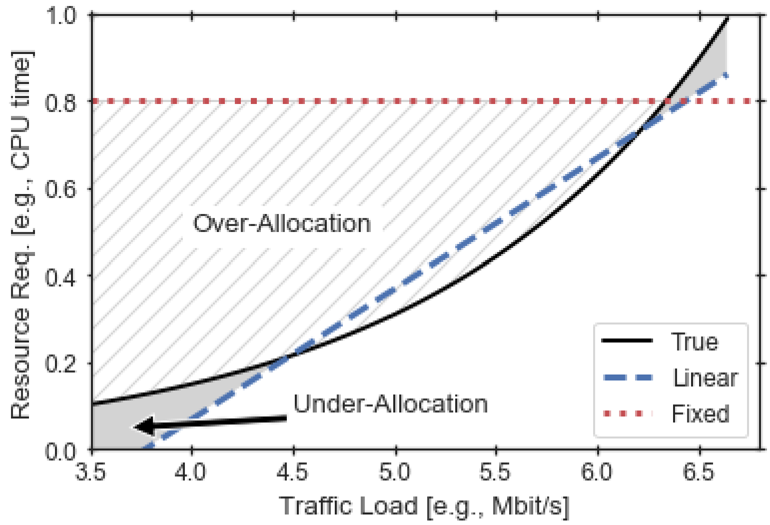
```
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

%matplotlib inline

sns.set(font_scale=1.1, style="white")
sns_cmap = sns.color_palette().as_hex()

cpu_col = 'Resource Req. [e.g., CPU time]'
thr_col = 'Traffic Load [e.g., Mbit/s]'
def gen_benchmark(cpu, coeff1=1, coeff2=1):
    return coeff1 * math.log2(1 + coeff2 * cpu)

def synthetic_benchmark():
    cpu_list = np.arange(0.1, 1, .01)
    data = []
    for cpu in cpu_list:
        data.append([cpu, gen_benchmark(cpu, coeff2=100)])
    return pd.DataFrame(data, columns=[cpu_col, thr_col])

def inverse(thru, coeff2=1):
    return (1/coeff2) * (2**thru - 1)
df = synthetic_benchmark()
# sns.lineplot(df[cpu_col], df[thr_col])
sns.lineplot(df[thr_col], df[cpu_col])

cpu = [inverse(thr, coeff2=100) for thr in df[thr_col]]
sns.lineplot(df[thr_col], cpu)
```




```
X = df[[thr_col]]
y = df[cpu_col]
lin_reg = LinearRegression()
lin_reg.fit(X, y)
lin_col = 'Linear'
df[lin_col] = lin_reg.predict(X)

joblib.dump(lin_reg, 'ml_models/synth_data/linear.joblib')
boost = GradientBoostingRegressor()
boost.fit(X, y)
boost_col = 'Boosting'
df[boost_col] = boost.predict(X)
joblib.dump(boost, 'ml_models/synth_data/boosting.joblib')

# nice fit
mean_squared_error(df[cpu_col], df[boost_col])
boost.predict([[10.0]])
inverse(10, coeff2=100)
# plotting for cpu 0-100
def plot_cpu_0_100():
    fig, ax = plt.subplots()

    fixed = 80

    # lines
    plt.plot(df[thr_col], df[cpu_col], label='True', color='black',
linewidth=2)
    plt.plot(df[thr_col], df[lin_col], color='blue', linewidth=2)
    plt.axhline(y=fixed, label='Fixed', color='green', linewidth=2)

    # fill in between
```

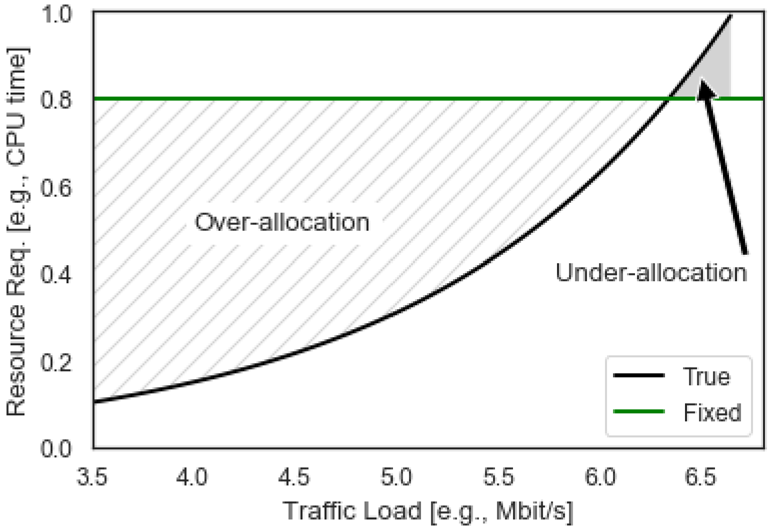
```
    ax.fill_between(df[thr_col], df[cpu_col], fixed, where=df[cpu_col]<fixed,
facecolor='white', edgecolor='grey', hatch='//')
    ax.fill_between(df[thr_col], df[lin_col], df[cpu_col],
where=df[lin_col]<df[cpu_col], facecolor='lightgrey', edgecolor='grey',
hatch='\\')

    # # text
    ax.text(10.5, 60, 'Over-allocation', bbox={'facecolor': 'white'})
    ax.annotate('Under-allocation', xy=(10.2, 5), xytext=(11, 5),
arrowprops={'facecolor': 'black'}, verticalalignment='center')

    plt.xlabel(thr_col)
    plt.ylabel(cpu_col)
    plt.xlim(10)
    #plt.ylim(0, 1)
    plt.legend()

    fig.savefig('plots/example_alloc.pdf', bbox_inches='tight')

# plot_cpu_0_100()
# plotting for cpu 0-1
fig, ax = plt.subplots()

fixed = 0.8

# lines
plt.plot(df[thr_col], df[cpu_col], label='True', color='black', linewidth=2)
plt.plot(df[thr_col], df[lin_col], color=sns_cmap[0], linewidth=3,
linestyle="dashed")
# plt.plot(df[thr_col], df[boost_col], color='red', linewidth=2)
plt.axhline(y=fixed, label='Fixed', color=sns_cmap[3], linewidth=3,
linestyle="dotted")

# fill in between
ax.fill_between(df[thr_col], df[cpu_col], fixed, where=df[cpu_col]<fixed,
facecolor='white', edgecolor='lightgrey', hatch='//')
ax.fill_between(df[thr_col], df[lin_col], df[cpu_col],
where=df[lin_col]<df[cpu_col], facecolor='lightgrey')

# # text
ax.text(4, 0.5, 'Over-Allocation', bbox={'facecolor': 'white'})
ax.annotate('Under-Allocation', xy=(3.7, 0.05), xytext=(4.5, 0.1),
arrowprops={'facecolor': 'black'}, verticalalignment='center')

plt.xlabel(thr_col)
plt.ylabel(cpu_col)
plt.xlim(3.5)
plt.ylim(0, 1)
```

```python
plt.tick_params(axis='both', direction='inout', length=5, bottom=True,
left=True, right=True, top=True)
plt.legend()

fig.savefig('plots/example_alloc.pdf', bbox_inches='tight')
# for presentation as png
fig.savefig('plots/example_alloc_lin.png', dpi=150, bbox_inches='tight')
```




```python
# 2nd: plot fixed resource alloc at 80%
fig, ax = plt.subplots()

fixed=0.8

# lines
plt.plot(df[thr_col], df[cpu_col], label='True', color='black', linewidth=2)
plt.axhline(y=fixed, label='Fixed', color='green', linewidth=2)

# fill in between
ax.fill_between(df[thr_col], df[cpu_col], fixed, where=df[cpu_col]<fixed,
facecolor='white', edgecolor='lightgrey', hatch='//')
ax.fill_between(df[thr_col], df[cpu_col], fixed, where=df[cpu_col]>fixed,
facecolor='lightgrey')

# # text
ax.text(4, 0.5, 'Over-allocation', bbox={'facecolor': 'white'})
ax.annotate('Under-allocation', xy=(6.5, 0.85), xytext=(6.25, 0.4),
arrowprops={'facecolor': 'black'}, verticalalignment='center',
horizontalalignment='center')

plt.xlabel(thr_col)
plt.ylabel(cpu_col)
plt.xlim(3.5)
plt.ylim(0, 1)
plt.legend()
```

```
fig.savefig('plots/example_alloc_fixed.png', dpi=150, bbox_inches='tight')
```



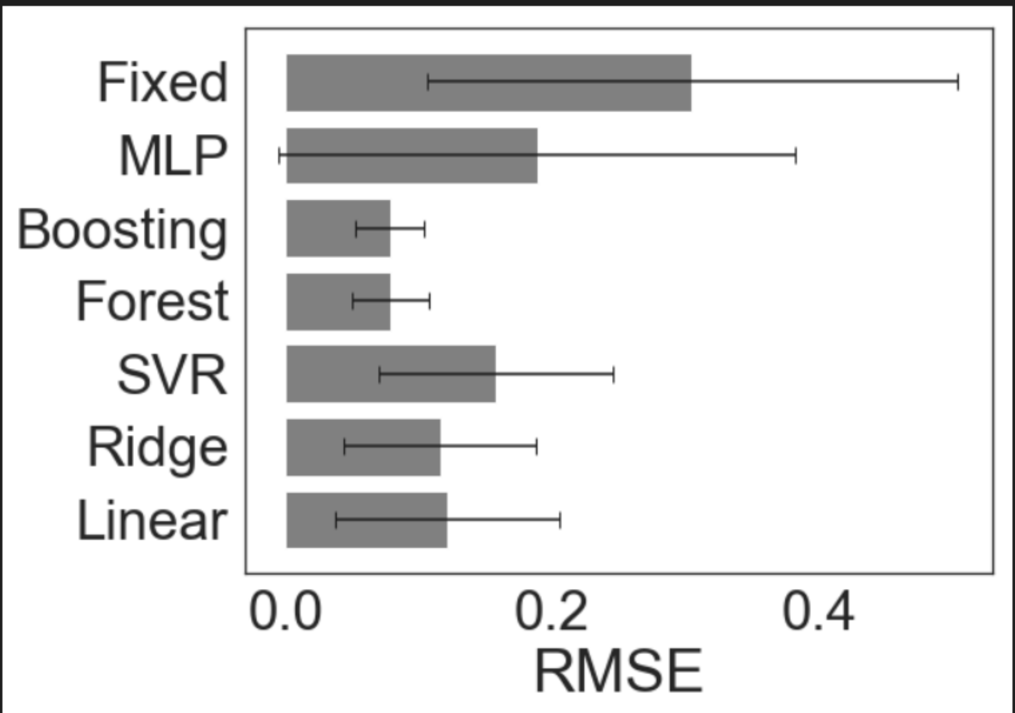
Script-2:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import os
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

from sklearn.base import BaseEstimator
from sklearn.model_selection import KFold, train_test_split, cross_val_score,
GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import LabelEncoder, StandardScaler,
PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

%matplotlib inline

sns.set(font_scale=3, style="white")
class FixedModel(BaseEstimator):
```

```python
    def __init__(self, fixed_value):
        self.fixed_value = fixed_value

    def fit(self, X, y):
        return self

    def predict(self, X):
        n_samples = X.shape[0]
        return [self.fixed_value for _ in range(n_samples)]
# define column names
cpu_col = 'Resource req. [e.g., CPU]'
thr_col = 'Traffic load [e.g., Mbit/s]'
# generate synthetic data
def gen_benchmark(cpu, coeff1=1, coeff2=1):
    return coeff1 * math.log2(1 + coeff2 * cpu)

def synthetic_benchmark():
    cpu_list = np.arange(0.1, 1, .01)
    data = []
    for cpu in cpu_list:
        data.append([cpu, gen_benchmark(cpu, coeff2=100)])
    return pd.DataFrame(data, columns=[cpu_col, thr_col])

# inverse function to calculate cpu given a throughput
def inverse(thru, coeff2=1):
    return (1/coeff2) * (2**thru - 1)
# prepare data
df = synthetic_benchmark()
X = df[[thr_col]]
y = df[cpu_col]
def cross_validation_rmse(model, X, y, k=5, save_model=False):
    scores = cross_val_score(model, X, y, scoring="neg_mean_squared_error",
cv=k)
    rmse = np.sqrt(-scores)
    name = type(model).__name__
    print(f"CV RMSE of {name}: {rmse.mean()} (+/-{rmse.std()})")
    if save_model:
        model.fit(X, y)
        joblib.dump(model, f'ml_models/synth_data/{name}.joblib')
    return rmse

def barplot_rmse(scores, labels, data_name):
    assert len(scores) == len(labels)
    rmse_mean = [s.mean() for s in scores]
    rmse_std = [s.std() for s in scores]
    x = np.arange(len(labels))

    fig, ax = plt.subplots(figsize = (8, 6))
```

```python
    plt.barh(x, rmse_mean, color='grey', xerr=rmse_std, capsize=5)

    ax.set_xlabel('RMSE')
    ax.set_yticks(x)
    ax.set_yticklabels(labels)

    fig.savefig(f'plots/{data_name}_rmse.pdf', bbox_inches='tight')
models = [LinearRegression(), Ridge(), SVR(), RandomForestRegressor(),
          GradientBoostingRegressor(), MLPRegressor(max_iter=1500),
          FixedModel(fixed_value=0.8)]
labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']
rmse = [cross_validation_rmse(model, X, y) for model in models]

barplot_rmse(rmse, labels, 'synth_default')
```

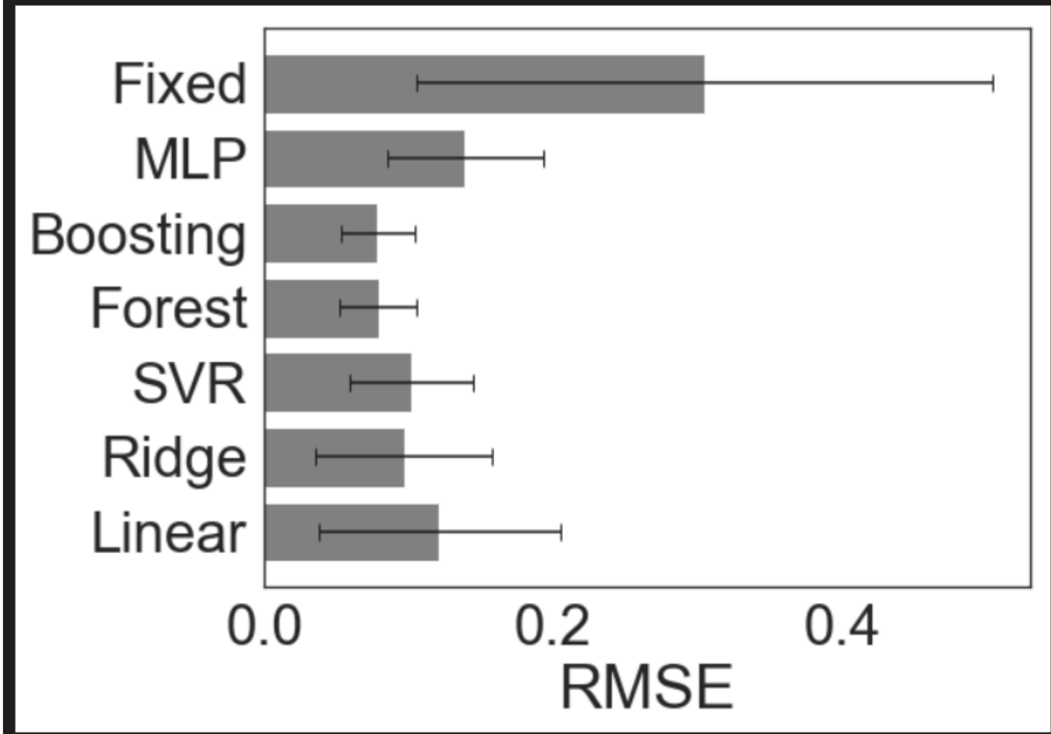
```
CV RMSE of LinearRegression: 0.12121162713679334 (+/-0.0834142420571489)
CV RMSE of Ridge: 0.1157083036553567 (+/-0.07204672163128789)
CV RMSE of SVR: 0.15772899390054107 (+/-0.08774146778153001)
CV RMSE of RandomForestRegressor: 0.079016930017029 (+/-0.02892004867091995)
CV RMSE of GradientBoostingRegressor: 0.07805364007331642 (+/-0.025433426403506574)
CV RMSE of MLPRegressor: 0.1884970322400931 (+/-0.19317890469508217)
CV RMSE of FixedModel: 0.30476112171679437 (+/-0.19909124881970067)
```




```python
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
os.makedirs(f'ml_models/synth_data', exist_ok=True)
joblib.dump(scaler, f'ml_models/synth_data/scaler.joblib')

rmse = [cross_validation_rmse(model, X_scaled, y) for model in models]
```

```
barplot_rmse(rmse, labels, 'synth_scaled')
```

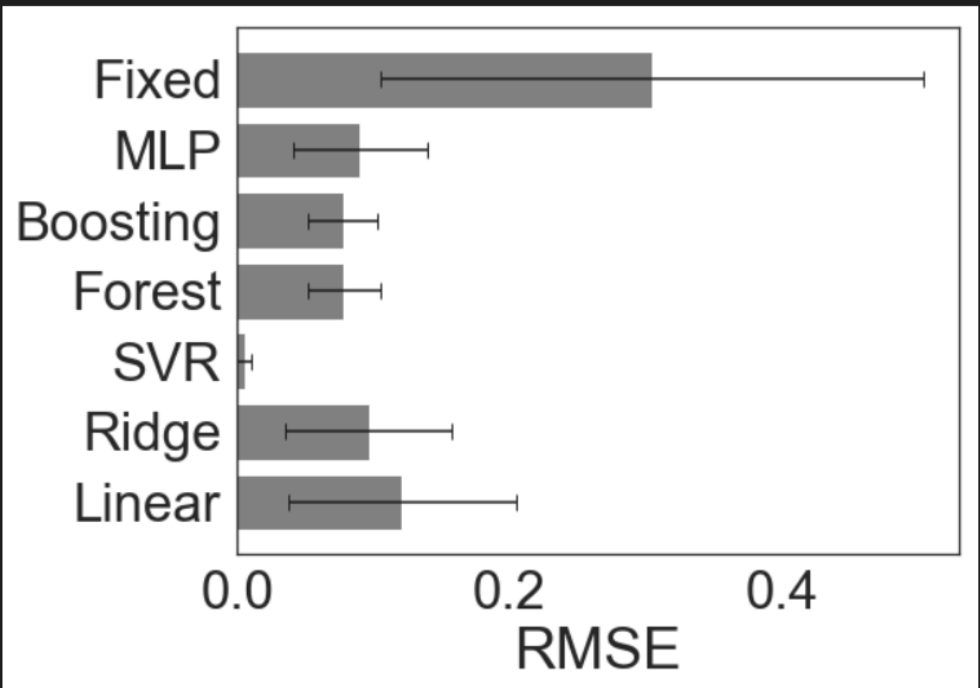
```
CV RMSE of LinearRegression: 0.12121162713679334 (+/-0.0834142420571489)
CV RMSE of Ridge: 0.09668457215749979 (+/-0.060822156542014665)
CV RMSE of SVR: 0.10148672105555254 (+/-0.04273273250016648)
CV RMSE of RandomForestRegressor: 0.07883837751219168 (+/-0.026953441506102894)
CV RMSE of GradientBoostingRegressor: 0.07805364007331642 (+/-0.025433426403506574)
CV RMSE of MLPRegressor: 0.1386704339625999 (+/-0.05389148824940066)
CV RMSE of FixedModel: 0.30476112171679437 (+/-0.19909124881970067)
```




```
def tune_hyperparams(model, X, y, params):
    grid_search = GridSearchCV(model, params, cv=5,
scoring="neg_mean_squared_error")
    grid_search.fit(X, y)
    return grid_search.best_estimator_
# hyperparam tuning
params_ridge = {'alpha': [0.1, 1, 10]}
params_svr = {'kernel': ['linear', 'poly', 'rbf'], 'C': [1, 10, 100],
              'epsilon': [0.001, 0.01, 0.1]}
params_forest = {'n_estimators': [10, 100, 200]}
params_boosting = {'learning_rate': [0.01, 0.1, 0.3], 'n_estimators': [10,
100, 200]}
params_mlp = {'hidden_layer_sizes': [(64,), (128,), (256)], 'alpha': [0.001,
0.0001, 0.00001],
              'learning_rate_init': [0.01, 0.001, 0.0001]}

params = [{}, params_ridge, params_svr, params_forest, params_boosting,
params_mlp, {}]
```

```
models_tuned = [tune_hyperparams(models[i], X_scaled, y, params[i]) for i in
range(len(models))]
rmse_tuned = [cross_validation_rmse(model, X_scaled, y, save_model=True) for
model in models_tuned]

barplot_rmse(rmse_tuned, labels, 'synth_tuned')
```

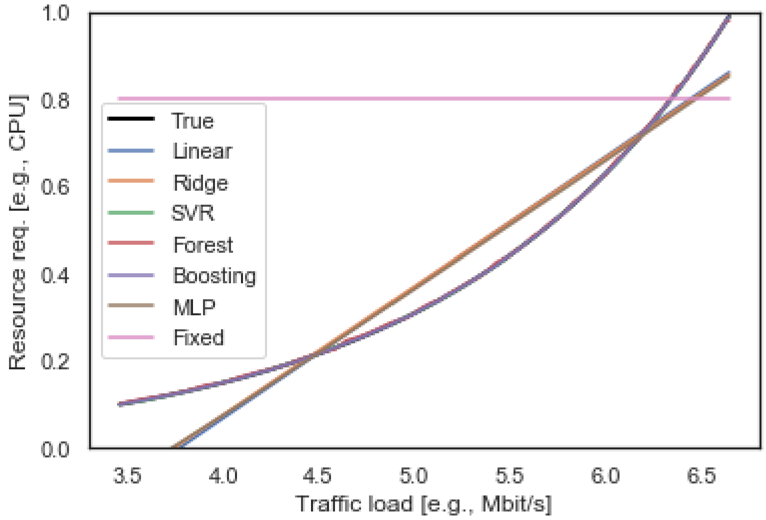
```
CV RMSE of LinearRegression: 0.12121162713679334 (+/-0.0834142420571489)
CV RMSE of Ridge: 0.09668457215749979 (+/-0.060822156542014665)
CV RMSE of SVR: 0.005448995192915848 (+/-0.005411929454877605)
CV RMSE of RandomForestRegressor: 0.07856988066678042 (+/-0.026776989848397992)
CV RMSE of GradientBoostingRegressor: 0.07755613024304038 (+/-0.025150165128635854)
CV RMSE of MLPRegressor: 0.09011693060882102 (+/-0.04934555264115149)
CV RMSE of FixedModel: 0.30476112171679437 (+/-0.19909124881970067)
```




```
# plot
sns.set(font_scale=1, style="white")

def plot_predictions(models, labels, X, y):
    assert len(models) == len(labels)

    fig, ax = plt.subplots()
    plt.plot(X[thr_col], y, label='True', color='black', linewidth=2)

    for i in range(len(models)):
        models[i].fit(X, y)
        plt.plot(X[thr_col], models[i].predict(X), label=labels[i])

    plt.xlabel(thr_col)
    plt.ylabel(cpu_col)
```

```
    plt.ylim(0, 1)
    plt.legend()
```

```
plot_predictions(models_tuned, labels, X, y)
```



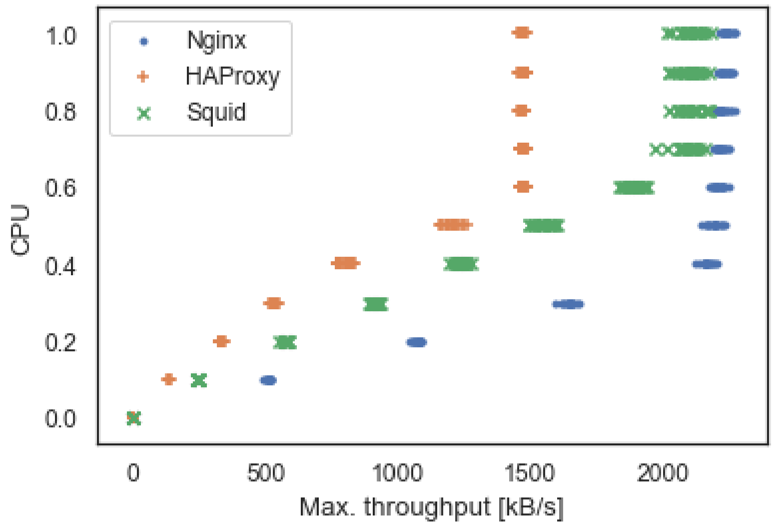
Script-3:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import time

from sklearn.base import BaseEstimator
from sklearn.model_selection import KFold, train_test_split, cross_val_score,
GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import LabelEncoder, StandardScaler,
PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

sns.set(font_scale=1.1, style='white')
class FixedModel(BaseEstimator):
```

```
    def __init__(self, fixed_value):
        self.fixed_value = fixed_value

    def fit(self, X, y):
        return self

    def predict(self, X):
        n_samples = X.shape[0]
        return [self.fixed_value for _ in range(n_samples)]
def select_and_rename(df, mapping):

    dff = df[list(mapping.keys())]
    # rename
    for k, v in mapping.items():
        dff.rename(columns={k: v}, inplace=True)
    return dff

def replaceSize(df):
    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e
/tngbench_share/ab_dist.csv -s 60 -k -i http://20.0.0.254:8888/", "small")
    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e
/tngbench_share/ab_dist.csv -s 60 -k http://20.0.0.254:8888/bunny.mp4", "big")
    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e
/tngbench_share/ab_dist.csv -s 60 -k -i -X 20.0.0.254:3128
http://40.0.0.254:80/", "small")
    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e
/tngbench_share/ab_dist.csv -s 60 -k -X 20.0.0.254:3128
http://40.0.0.254:80/bunny.mp4", "big")
    return df
# Load data from path
web1 = pd.read_csv("vnf_data/csv_experiments_WEB1.csv")
web2 = pd.read_csv("vnf_data/csv_experiments_WEB2.csv")
web3 = pd.read_csv("vnf_data/csv_experiments_WEB3.csv")

# do processing, renaming and selection
mapping = {
    "param__func__mp.input__cmd_start": "size",
    "metric__mp.input.vdu01.0__ab_transfer_rate_kbyte_per_second": "Max.
throughput [kB/s]",
}

mapping01 = mapping.copy()
mapping01["param__func__de.upb.lb-nginx.0.1__cpu_bw"] = "CPU"
mapping01["param__func__de.upb.lb-nginx.0.1__mem_max"] = "Memory"

mapping02 = mapping.copy()
mapping02["param__func__de.upb.lb-haproxy.0.1__cpu_bw"] = "CPU"
```

```
mapping02["param__func__de.upb.lb-haproxy.0.1__mem_max"] = "Memory"

mapping03 = mapping.copy()
mapping03["param__func__de.upb.px-squid.0.1__cpu_bw"] = "CPU"
mapping03["param__func__de.upb.px-squid.0.1__mem_max"] = "Memory"

web1 = select_and_rename(web1, mapping01)
web2 = select_and_rename(web2, mapping02)
web3 = select_and_rename(web3, mapping03)

web1 = replaceSize(web1)
web2 = replaceSize(web2)
web3 = replaceSize(web3)

mem = 128

web1_small = web1.loc[(web1["size"] == "small") & (web1["Memory"] == mem)]
web1_small = web1_small[["Max. throughput [kB/s]", "CPU"]]
web1_big = web1.loc[(web1["size"] == "big")  & (web1["Memory"] == mem)]
web1_big = web1_big[["Max. throughput [kB/s]", "CPU"]]

web2_small = web2.loc[(web2["size"] == "small")  & (web2["Memory"] == mem)]
web2_small = web2_small[["Max. throughput [kB/s]", "CPU"]]
web2_big = web2.loc[(web2["size"] == "big")  & (web2["Memory"] == mem)]
web2_big = web2_big[["Max. throughput [kB/s]", "CPU"]]

web3_small = web3.loc[(web3["size"] == "small")  & (web3["Memory"] == mem)]
web3_small = web3_small[["Max. throughput [kB/s]", "CPU"]]
web3_big = web3.loc[(web3["size"] == "big")  & (web3["Memory"] == mem)]
web3_big = web3_big[["Max. throughput [kB/s]", "CPU"]]
num_measures = 20
measures = [0 for _ in range(num_measures)]

web1_small = web1_small.append(pd.DataFrame({'Max. throughput [kB/s]':
measures, 'CPU': measures}), ignore_index=True)
web1_big = web1_big.append(pd.DataFrame({'Max. throughput [kB/s]': measures,
'CPU': measures}), ignore_index=True)
web2_small = web2_small.append(pd.DataFrame({'Max. throughput [kB/s]':
measures, 'CPU': measures}), ignore_index=True)
web2_big = web2_big.append(pd.DataFrame({'Max. throughput [kB/s]': measures,
'CPU': measures}), ignore_index=True)
web3_small = web3_small.append(pd.DataFrame({'Max. throughput [kB/s]':
measures, 'CPU': measures}), ignore_index=True)
web3_big = web3_big.append(pd.DataFrame({'Max. throughput [kB/s]': measures,
'CPU': measures}), ignore_index=True)
nginx = web1_small
haproxy = web2_small
squid = web3_small
```

```
def plot_vnf_data():
    sns.set(font_scale=1.1, style='white')
    fig, ax = plt.subplots()
    plt.scatter(nginx['Max. throughput [kB/s]'], nginx['CPU'], label='Nginx',
marker='.')
    plt.scatter(haproxy['Max. throughput [kB/s]'], haproxy['CPU'],
label='HAProxy', marker='+')
    plt.scatter(squid['Max. throughput [kB/s]'], squid['CPU'], label='Squid',
marker='x')

    # labels
    ax.set_xlabel('Max. throughput [kB/s]')
    ax.set_ylabel('CPU')
    plt.legend()

    fig.savefig(f'plots/web_vnf_data.pdf', bbox_inches='tight')

plot_vnf_data()
```




```
def cross_validation_rmse(model, X, y, vnf_name, k=5, save_model=False):
    scores = cross_val_score(model, X, y, scoring="neg_mean_squared_error",
cv=k)
    rmse = np.sqrt(-scores)
    name = type(model).__name__
    print(f"CV RMSE of {name}: {rmse.mean()} (+/-{rmse.std()})")
    if save_model:
        model.fit(X, y)
        joblib.dump(model, f'ml_models/{vnf_name}/{name}.joblib')
    return rmse

def tune_hyperparams(model, X, y, params):
    grid_search = GridSearchCV(model, params, cv=5,
scoring="neg_mean_squared_error")
```

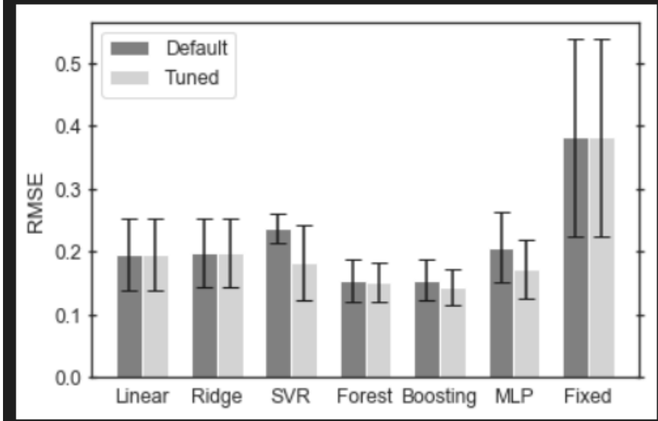
```
    grid_search.fit(X, y)
    return grid_search.best_estimator_

def barplot_rmse(scores, labels, data_name):
    sns.set(font_scale=1.1, style='white')
    assert len(scores) == len(labels)

    # preparation
    rmse_mean = [s.mean() for s in scores]
    rmse_std = [s.std() for s in scores]
    x = np.arange(len(labels))

    # plot
    fig, ax = plt.subplots() #figsize = (8, 6))
    plt.barh(x, rmse_mean, color='grey', xerr=rmse_std, capsize=5)

    # labels
    ax.set_xlabel('RMSE')
    ax.set_yticks(x)
    ax.set_yticklabels(labels)

    fig.savefig(f'plots/{data_name}_rmse.pdf', bbox_inches='tight')
def train_eval_models(X, y, vnf_name, tune_params=False):
    # prepare models
    labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']
    models = [LinearRegression(), Ridge(), SVR(), RandomForestRegressor(),
              GradientBoostingRegressor(), MLPRegressor(max_iter=1500),
              FixedModel(fixed_value=0.8)]

    # params for tuning
    params_ridge = {'alpha': [0.1, 1, 10]}
    params_svr = {'kernel': ['linear', 'poly', 'rbf'], 'C': [1, 10, 100],
                  'epsilon': [0.001, 0.01, 0.1]}
    params_forest = {'n_estimators': [10, 100, 200]}
    params_boosting = {'learning_rate': [0.01, 0.1, 0.3], 'n_estimators': [10,
100, 200]}
    params_mlp = {'hidden_layer_sizes': [(64,), (128,), (256)], 'alpha':
[0.001, 0.0001, 0.00001],
                  'learning_rate_init': [0.01, 0.001, 0.0001]}
    params = [{}, params_ridge, params_svr, params_forest, params_boosting,
params_mlp, {}]

    # tune, train, eval
    if tune_params:
        models = [tune_hyperparams(models[i], X, y, params[i]) for i in
range(len(models))]
    rmse = [cross_validation_rmse(model, X, y, vnf_name, save_model=True) for
model in models]
```

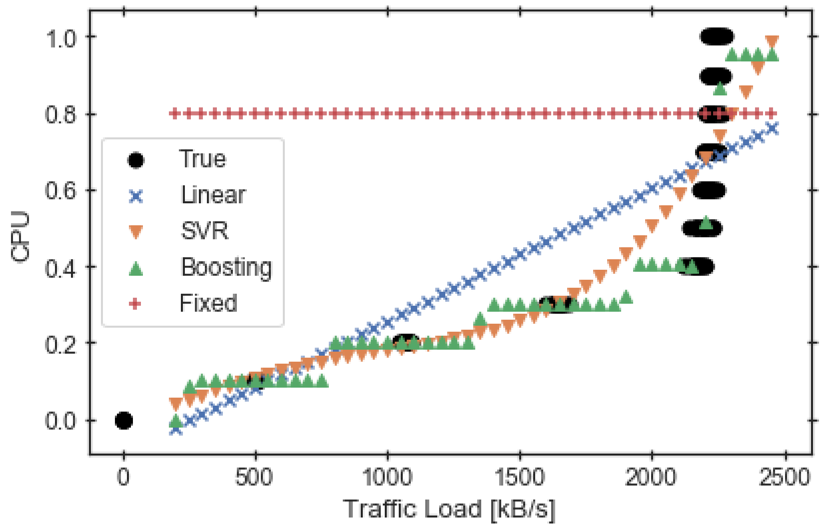
```
    # plot
    if tune_params:
        barplot_rmse(rmse, labels, f'{vnf_name}_tuned')
    else:
        barplot_rmse(rmse, labels, f'{vnf_name}_default')

    return models
def prepare_data(data, vnf_name):
    X = data[['Max. throughput [kB/s]']]
    y = data['CPU']
    X = X.fillna(X.median())

    scaler = MinMaxScaler()
    scaler.fit(X)
    os.makedirs(f'ml_models/{vnf_name}', exist_ok=True)
    joblib.dump(scaler, f'ml_models/{vnf_name}/scaler.joblib')

    return X, y, scaler
def predict_plot_all(models, scaler, X, y, vnf_name):
    sns.set(font_scale=1.1, style='white')

    models = [models[0], models[2], models[3], models[6]]
    labels = ['Linear', 'SVR', 'Boosting', 'Fixed']
    markers = ['x', 'v', '^', '+']
    colors = ['blue', 'orange', 'red', 'green']

    fig, ax = plt.subplots()
    plt.scatter(X, y, label='True', marker='o', color='black', s=50)
    X = scaler.transform(X)
    times = []
    for i, model in enumerate(models):
        name = type(model).__name__
        model.fit(X, y)
        os.makedirs(f'ml_models/{vnf_name}', exist_ok=True)
        X_plot = pd.DataFrame({'Max. Throughput [kB/s]': np.arange(200, 2500,
50)})
        X_plot_scaled = scaler.transform(X_plot)
        start = time.time()
        y_pred = model.predict(X_plot_scaled)
        times.append(start - time.time())
        plt.scatter(X_plot, y_pred, label=labels[i], marker=markers[i])

    plt.xlabel('Traffic Load [kB/s]')
    plt.ylabel('CPU')
    plt.tick_params(axis='both', direction='inout', length=5, bottom=True,
left=True, right=True, top=True)
    plt.legend()
```

```
    # save, avoid cutting off labels
    plt.tight_layout()
    fig.savefig(f'plots/{vnf_name}_model_comparison.pdf')

    return times
def barplot_compare_rmse(scores_default, scores_tuned, labels, data_name):

    sns.set(font_scale=1.1, style='white')
    assert len(scores_default) == len(scores_tuned) == len(labels)

    x = np.arange(len(labels))
    width = 0.35

    # plot
    fig, ax = plt.subplots() # prev: figsize = (8, 5))

    # default
    rmse_mean = [s.mean() for s in scores_default]
    rmse_std = [s.std() for s in scores_default]
    ci95 = [1.96 * std / np.sqrt(len(rmse_std)) for std in rmse_std]
    ax.bar(x - width/2, rmse_mean, width, yerr=ci95, capsize=5, color='gray',
label='Default')

    # same for tuned version
    rmse_mean = [s.mean() for s in scores_tuned]
    rmse_std = [s.std() for s in scores_tuned]
    ci95 = [1.96 * std / np.sqrt(len(rmse_std)) for std in rmse_std]
    ax.bar(x + width/2, rmse_mean, width, yerr=ci95, capsize=5,
color='lightgray', label='Tuned')

    # labels
    ax.set_ylabel('RMSE')
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.tick_params(axis='both', direction='inout', length=5, bottom=False,
left=True, right=True, top=False)
    ax.legend()

    fig.savefig(f'plots/{data_name}_rmse.pdf', bbox_inches='tight')
def train_tune_eval_models(X, y, vnf_name):
    # prepare models and rmse without tuning
    labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']
    models_default = [LinearRegression(), Ridge(), SVR(),
RandomForestRegressor(),
              GradientBoostingRegressor(), MLPRegressor(max_iter=1500),
              FixedModel(fixed_value=0.8)]
    rmse_default = [cross_validation_rmse(model, X, y, vnf_name,
save_model=False) for model in models_default]
```

```
    # hyperparam tuning
    params_ridge = {'alpha': [0.1, 1, 10]}
    params_svr = {'kernel': ['poly', 'rbf'], 'C': [1, 10, 100],
                  'epsilon': [0.001, 0.01, 0.1]}
    params_forest = {'n_estimators': [10, 100, 200]}
    params_boosting = {'learning_rate': [0.01, 0.1, 0.3], 'n_estimators': [10,
100, 200]}
    params_mlp = {'hidden_layer_sizes': [(64,), (128,), (256)], 'alpha':
[0.001, 0.0001, 0.00001],
                  'learning_rate_init': [0.01, 0.001, 0.0001]}
    params = [{}, params_ridge, params_svr, params_forest, params_boosting,
params_mlp, {}]

    models_tuned = [tune_hyperparams(models_default[i], X, y, params[i]) for i
in range(len(labels))]
    rmse_tuned = [cross_validation_rmse(model, X, y, vnf_name,
save_model=True) for model in models_tuned]

    barplot_compare_rmse(rmse_default, rmse_tuned, labels,
f'{vnf_name}_default-tuned')

    return models_tuned
vnf_name = 'nginx'
X, y, scaler = prepare_data(nginx, vnf_name)
X_scaled = scaler.transform(X)

models = train_tune_eval_models(X_scaled, y, vnf_name)
predict_plot_all(models, scaler, X, y, vnf_name)
```

```
CV RMSE of LinearRegression: 0.19651294327683955 (+/-0.0770302015503735)
CV RMSE of Ridge: 0.19747963738692514 (+/-0.07374557483743582)
CV RMSE of SVR: 0.23843849102511663 (+/-0.03133600651863603)
CV RMSE of RandomForestRegressor: 0.1544186506720129 (+/-0.04566173409977426)
CV RMSE of GradientBoostingRegressor: 0.15532875692425963 (+/-0.04397880798814986)
CV RMSE of MLPRegressor: 0.20690428180788495 (+/-0.07598079094190109)
CV RMSE of FixedModel: 0.38153965300099446 (+/-0.21077830340877302)
CV RMSE of LinearRegression: 0.19651294327683955 (+/-0.0770302015503735)
CV RMSE of Ridge: 0.19747963738692514 (+/-0.07374557483743582)
CV RMSE of SVR: 0.1829420132476261 (+/-0.08049356087609329)
CV RMSE of RandomForestRegressor: 0.1523560777807378 (+/-0.04203026966747045)
CV RMSE of GradientBoostingRegressor: 0.14334186263227305 (+/-0.0379499323771591)
CV RMSE of MLPRegressor: 0.17355377473628966 (+/-0.06294255153363573)
CV RMSE of FixedModel: 0.38153965300099446 (+/-0.21077830340877302)

[0.0, 0.0, -0.015516519546508789, 0.0]
```



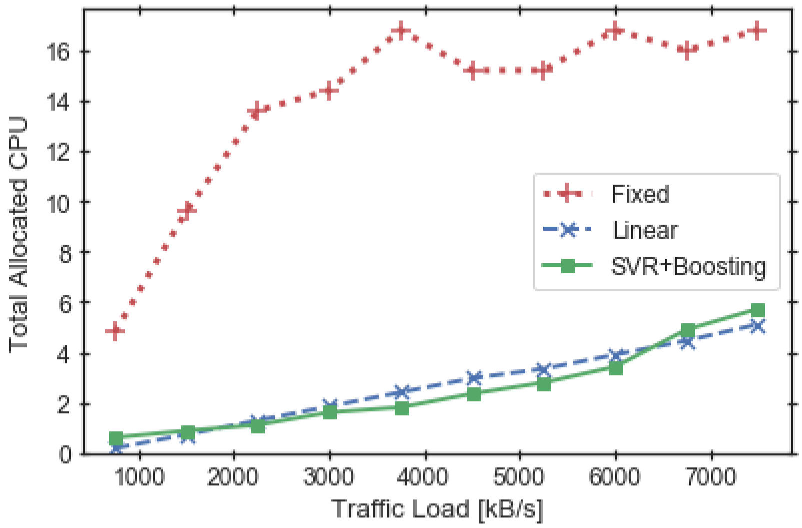


Script-4:

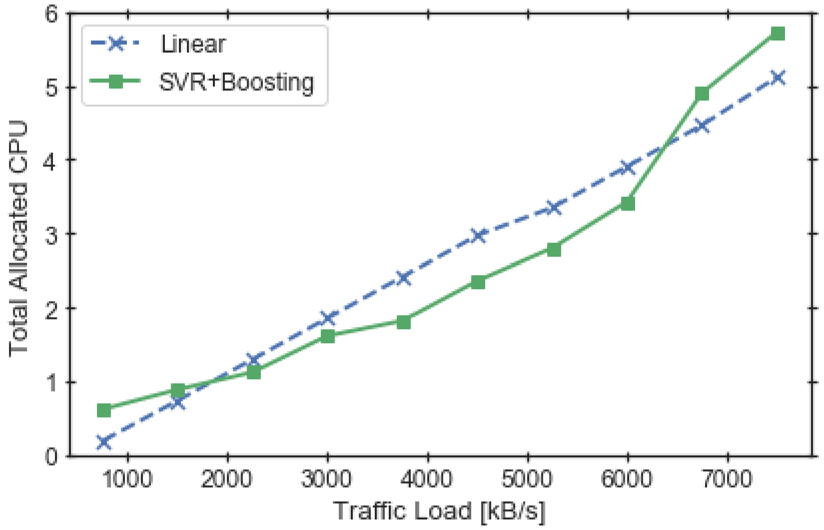
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```python
import matplotlib as mpl
import seaborn as sns
import sklearn
import yaml
import sys
import glob
import os
%matplotlib inline
sns.set(font_scale=1.1, style='white')
sns_cmap = sns.color_palette().as_hex()
def sum_cpu(node_res):
    cpu = sum([v['cpu'] for v in node_res])
    return cpu

def read_placement(placement, df_data, flow_dr=250):
    df_data['num_flows'].append(placement['input']['num_flows'])
    df_data['num_sources'].append(placement['input']['num_sources'])
    df_data['source_dr'].append(placement['input']['num_flows'] * flow_dr)
    df_data['num_instances'].append(placement['metrics']['num_instances'])
    df_data['max_e2e_delay'].append(placement['metrics']['max_endToEnd_delay']
)
    df_data['total_delay'].append(placement['metrics']['total_delay'])
    df_data['runtime'].append(placement['metrics']['runtime'])
    df_data['total_cpu'].append(sum_cpu(placement['placement']['alloc_node_res
']))
    return df_data

def read_results(results):
    data = {'num_sources' : [], 'num_flows': [], 'source_dr': [],
'num_instances': [],
            'max_e2e_delay': [], 'total_delay': [], 'runtime': [],
'total_cpu': []}

    # iterate through result files
    for res in glob.glob(results):
        # open and save metrics of interest
        with open(res, 'r') as f:
            placement = yaml.load(f, Loader=yaml.SafeLoader)
            data = read_placement(placement, data)

    return pd.DataFrame(data).sort_values(by=['num_flows'])
# read results
dataset = 'web_data'
sources = 'three_source_dr250'
results = f'placement_data/{dataset}/{sources}/'

df_true = read_results(results + 'true/*.yaml')
df_fixed = read_results(results + 'fixed/*.yaml')
```

```python
df_linear = read_results(results + 'linear/*.yaml')
df_boost = read_results(results + 'boosting/*.yaml')
df_svr = read_results(results + 'svr/*.yaml')
df_ml = read_results(results + 'ml/*.yaml')
def plot(x_col, x_label, y_col, y_label, save_plot=True, plot_fixed=True):
    sns.set(font_scale=1.1, style='white')
    fig, ax = plt.subplots()

    if plot_fixed:
        plt.plot(df_fixed[x_col], df_fixed[y_col], label='Fixed',
color=sns_cmap[3], marker='+',
                 linewidth=3, linestyle="dotted", markersize=10,
markeredgewidth=1.5)
    plt.plot(df_linear[x_col], df_linear[y_col], label='Linear',
color=sns_cmap[0], marker='x',
             linewidth=2, linestyle="dashed", markersize=7,
markeredgewidth=1.5)
    plt.plot(df_ml[x_col], df_ml[y_col], label='SVR+Boosting',
color=sns_cmap[2], marker='s',
             linewidth=2)

    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.ylim(bottom=0)
    plt.tick_params(axis='both', direction='inout', length=5, bottom=True,
left=True, right=True, top=True)
    plt.legend()

    plt.tight_layout()
    if save_plot:
        fig.savefig(f'plots/{dataset}_{y_col}.pdf')

    return fig, ax
plot('source_dr', 'Traffic Load [kB/s]', 'total_cpu', 'Total Allocated CPU')
```

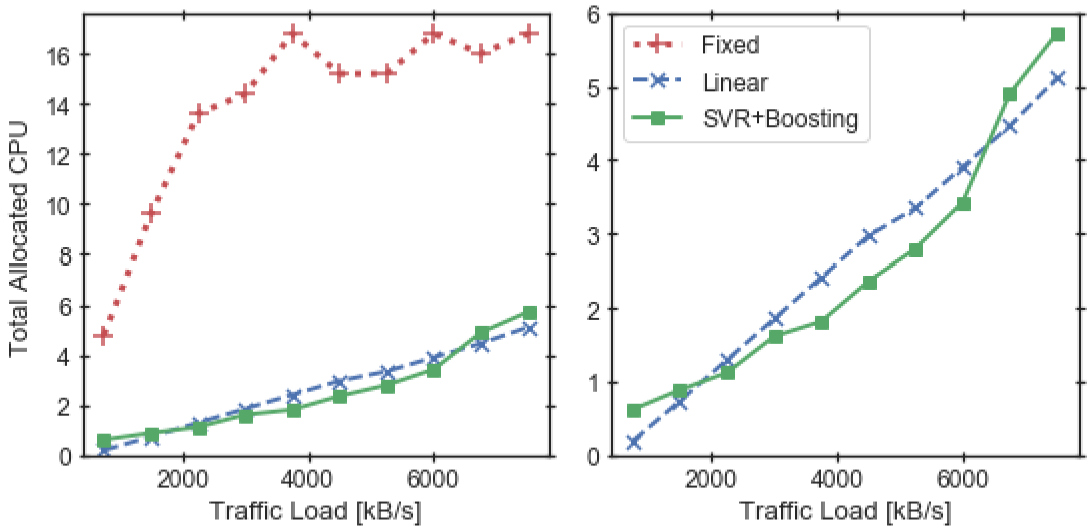


```
fig, ax = plot('source_dr', 'Traffic Load [kB/s]', 'total_cpu', 'Total
Allocated CPU', plot_fixed=False)
```

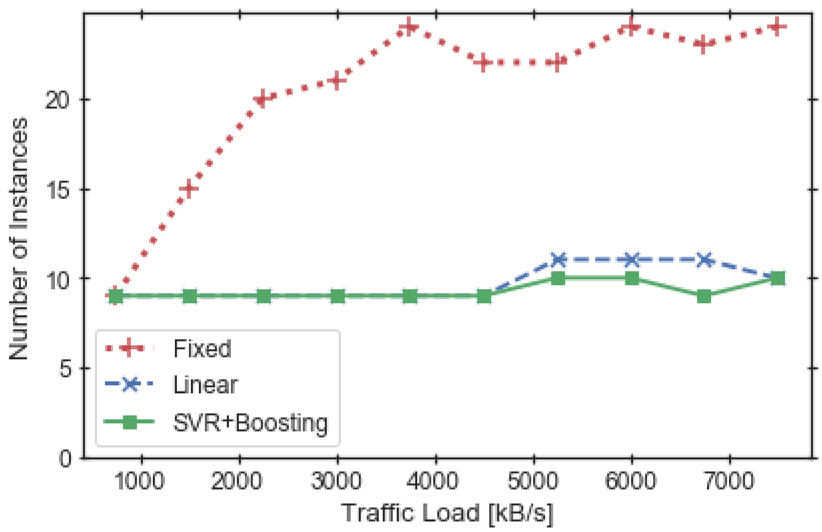



```
# plot both
def plot_both(x_col, x_label, y_col, y_label):
    sns.set(font_scale=1.1, style='white')
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

    ax[0].plot(df_fixed[x_col], df_fixed[y_col], label='Fixed',
color=sns_cmap[3], marker='+',
               linewidth=3, linestyle="dotted", markersize=10,
markeredgewidth=1.5)
    ax[0].plot(df_linear[x_col], df_linear[y_col], label='Linear',
color=sns_cmap[0], marker='x',
               linewidth=2, linestyle="dashed", markersize=7,
markeredgewidth=1.5)
    ax[0].plot(df_ml[x_col], df_ml[y_col], label='SVR+Boosting',
color=sns_cmap[2], marker='s',
               linewidth=2)
    ax[0].set(xlabel=x_label, ylabel=y_label)
    ax[0].set_ylim(bottom=0)
    ax[0].tick_params(axis='both', direction='inout', length=5, bottom=True,
left=True, right=True, top=True)

#     ax[0].legend()

    ax[1].plot([], [], label='Fixed', color=sns_cmap[3], marker='+',
linewidth=3,
               linestyle="dotted", markersize=10, markeredgewidth=1.5)
    ax[1].plot(df_linear[x_col], df_linear[y_col], label='Linear',
color=sns_cmap[0], marker='x',
               linewidth=2, linestyle="dashed", markersize=7,
markeredgewidth=1.5)
```

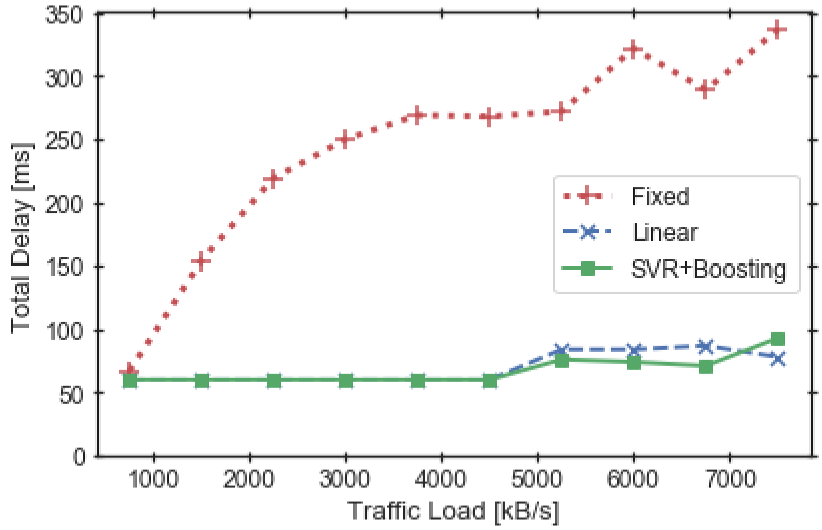
```
    ax[1].plot(df_ml[x_col], df_ml[y_col], label='SVR+Boosting',
color=sns_cmap[2], marker='s',
               linewidth=2)
    ax[1].set(xlabel=x_label)
    ax[1].set_ylim(bottom=0)
    ax[1].tick_params(axis='both', direction='inout', length=5, bottom=True,
left=True, right=True, top=True)
    ax[1].legend()

    # avoid cutting off figure labels
    plt.tight_layout()
    fig.savefig(f'plots/{dataset}_{y_col}_both.pdf')

plot_both('source_dr', 'Traffic Load [kB/s]', 'total_cpu', 'Total Allocated
CPU')
```



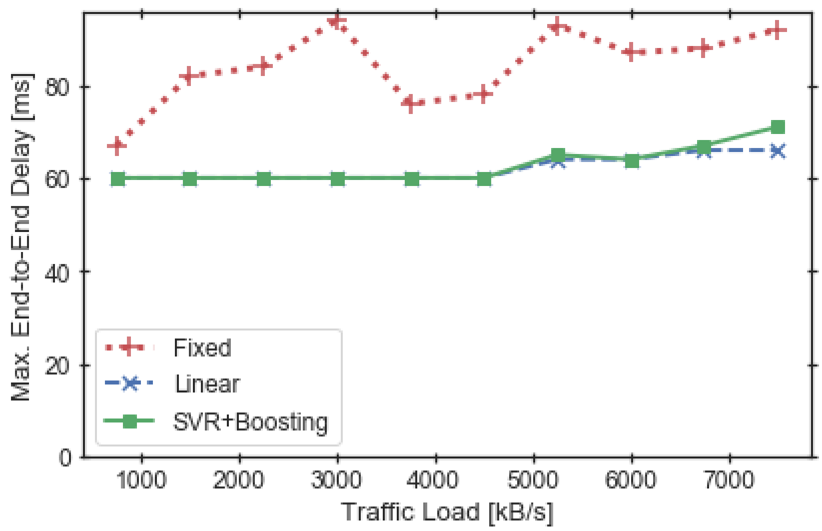

```
plot('source_dr', 'Traffic Load [kB/s]', 'num_instances', 'Number of
Instances')
```



```
plot('source_dr', 'Traffic Load [kB/s]', 'total_delay', 'Total Delay [ms]')
```




```
plot('source_dr', 'Traffic Load [kB/s]', 'max_e2e_delay', 'Max. End-to-End
Delay [ms]')
```



Script-5:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
```

```python
import random
import glob
import yaml
import timeit
import math

from sklearn.base import BaseEstimator
from sklearn.model_selection import KFold, train_test_split, cross_val_score,
GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import LabelEncoder, StandardScaler,
PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error


sns.set(font_scale=1.1, style='white')
```

```python
class FixedModel(BaseEstimator):

    def __init__(self, fixed_value):
        self.fixed_value = fixed_value

    def fit(self, X, y):
        return self

    def predict(self, X):
        n_samples = X.shape[0]
        return [self.fixed_value for _ in range(n_samples)]
# function for processing and simplifying the dataset
def select_and_rename(df, mapping):

    # select subset of columns
    dff = df[list(mapping.keys())]
    # rename
    for k, v in mapping.items():
        dff.rename(columns={k: v}, inplace=True)
    return dff

def replaceSize(df):
    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e
/tngbench_share/ab_dist.csv -s 60 -k -i http://20.0.0.254:8888/", "small")
    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e
/tngbench_share/ab_dist.csv -s 60 -k http://20.0.0.254:8888/bunny.mp4", "big")
```

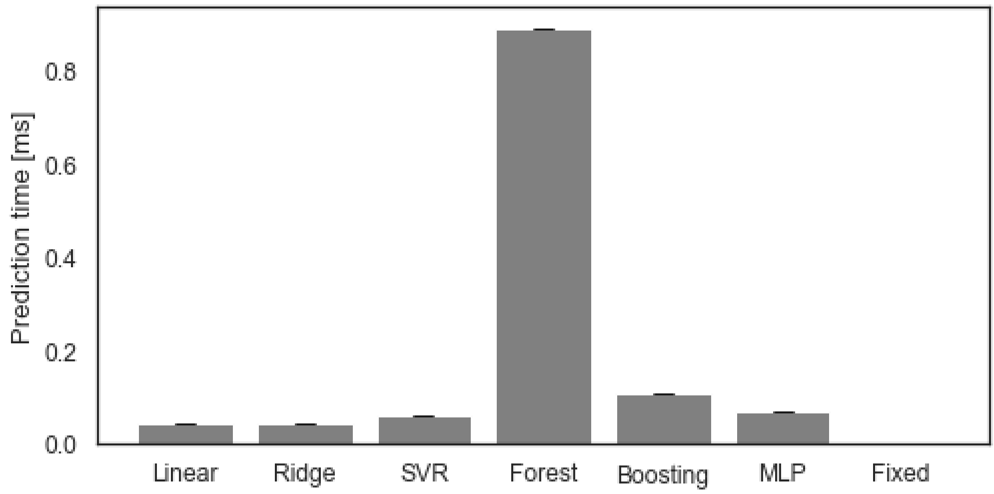
```
    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e
/tngbench_share/ab_dist.csv -s 60 -k -i -X 20.0.0.254:3128
http://40.0.0.254:80/", "small")
    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e
/tngbench_share/ab_dist.csv -s 60 -k -X 20.0.0.254:3128
http://40.0.0.254:80/bunny.mp4", "big")
    return df

web1 = pd.read_csv("vnf_data/csv_experiments_WEB1.csv")
web3 = pd.read_csv("vnf_data/csv_experiments_WEB3.csv")

mapping = {
    "param__func__mp.input__cmd_start": "size",
    "metric__mp.input.vdu01.0__ab_transfer_rate_kbyte_per_second": "Max.
throughput [kB/s]",
}

mapping01 = mapping.copy()
mapping01["param__func__de.upb.lb-nginx.0.1__cpu_bw"] = "CPU"
mapping01["param__func__de.upb.lb-nginx.0.1__mem_max"] = "Memory"

mapping03 = mapping.copy()
mapping03["param__func__de.upb.px-squid.0.1__cpu_bw"] = "CPU"
mapping03["param__func__de.upb.px-squid.0.1__mem_max"] = "Memory"

web1 = select_and_rename(web1, mapping01)
web3 = select_and_rename(web3, mapping03)

web1 = replaceSize(web1)
web3 = replaceSize(web3)
# select sub-datasets with small and large flows
# and with specific memory
mem = 128

web1_small = web1.loc[(web1["size"] == "small") & (web1["Memory"] == mem)]
web1_small = web1_small[["Max. throughput [kB/s]", "CPU"]]
web3_small = web3.loc[(web3["size"] == "small")  & (web3["Memory"] == mem)]
web3_small = web3_small[["Max. throughput [kB/s]", "CPU"]]

# add 20 "measurements" at 0 CPU and throuhgput
num_measures = 20
measures = [0 for _ in range(num_measures)]
web1_small = web1_small.append(pd.DataFrame({'Max. throughput [kB/s]':
measures, 'CPU': measures}), ignore_index=True)
web3_small = web3_small.append(pd.DataFrame({'Max. throughput [kB/s]':
measures, 'CPU': measures}), ignore_index=True)
def prepare_data(data):
    X = data[['Max. throughput [kB/s]']]
```

```python
    y = data['CPU']
    X = X.fillna(X.median())

    scaler = MinMaxScaler()
    scaler.fit_transform(X)

    return X, y
def barplot(times, labels, ylabel='Time [s]', filename=None):
    assert len(times) == len(labels)

    times_mean = [np.array(t).mean() for t in times]
    print(times_mean)
    times_std = [np.array(t).std() for t in times]
    x = np.arange(len(labels))

    sns.set(font_scale=1.1, style='white')
    fig, ax = plt.subplots(figsize=(8, 4))
    plt.bar(x, times_mean, yerr=times_std, capsize=5, color='gray')

    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.set_ylabel(ylabel)

    if filename is not None:
        fig.savefig(f'plots/{filename}.pdf', bbox_inches='tight')
# prepare data
X_nginx, y_nginx = prepare_data(web1_small)
X_squid, y_squid = prepare_data(web3_small)

X = X_nginx
y = y_nginx
vnf_name = 'nginx'
# generate synthetic data
def gen_benchmark(cpu, coeff1=1, coeff2=1):
    return coeff1 * math.log2(1 + coeff2 * cpu)

def synthetic_benchmark(n):
    cpu_list = [random.random() for _ in range(n)]
    thr_list = [gen_benchmark(cpu, coeff2=100) for cpu in cpu_list]
    X = pd.DataFrame(data={'Throughput': thr_list})
    y = np.array(cpu_list)
    return X, y
def training_times(models, X, y, scaler):
    # measure training times
    times = []
    for model in models:
        print(type(model).__name__)
        X_scaled = scaler.transform(X)
```

```
        t = %timeit -o model.fit(X, y)
        times.append(t)
    return times
labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']
models = [LinearRegression(), Ridge(), SVR(verbose=True),
RandomForestRegressor(n_estimators=10),
          GradientBoostingRegressor(), MLPRegressor(max_iter=1500),
          FixedModel(fixed_value=0.8)]
model_names = [type(model).__name__ for model in models]
models = [joblib.load(f'ml_models/{vnf_name}/{name}.joblib') for name in
model_names]
scaler = joblib.load(f'ml_models/{vnf_name}/scaler.joblib')

models = [joblib.load(f'ml_models/{vnf_name}/{name}.joblib') for name in
model_names]
scaler = joblib.load(f'ml_models/{vnf_name}/scaler.joblib')
X_rand = pd.DataFrame(data={'Rand max. throughput': [random.randrange(0, 3000)
for _ in range(1)]})
times = []
for model in models:
    print(type(model).__name__)
    X_scaled = scaler.transform(X_rand)
    t = %timeit -o model.predict(X_scaled)
    times.append(t)
```

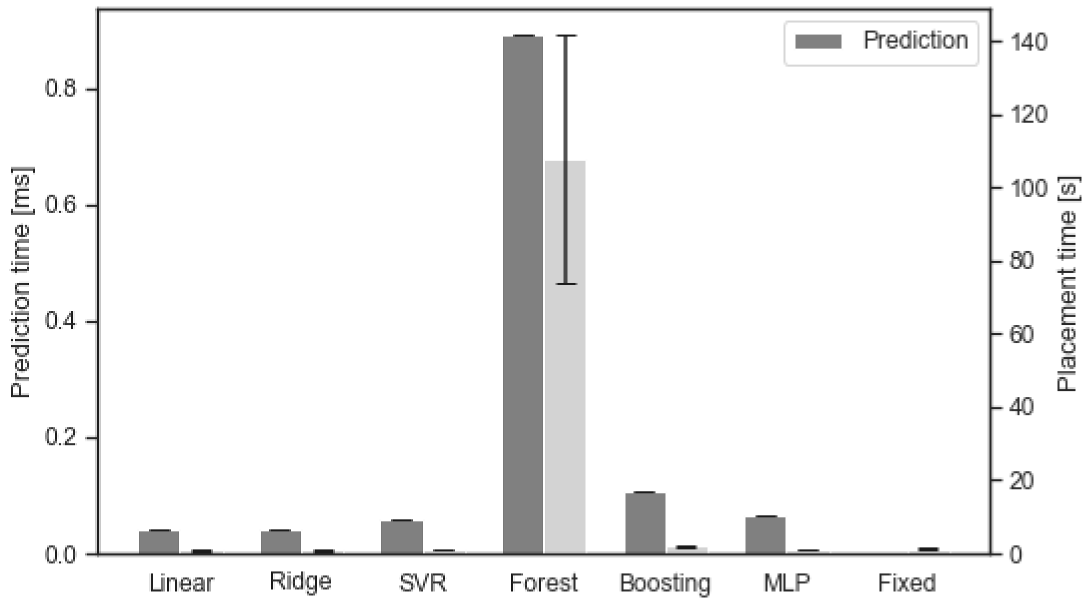
```
LinearRegression
The slowest run took 4.49 times longer than the fastest. This could mean that an intermediate result is being cached.
83.2 µs ± 63.5 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
Ridge
64.5 µs ± 17.2 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
SVR
86.2 µs ± 19 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
RandomForestRegressor
7.29 ms ± 170 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
GradientBoostingRegressor
123 µs ± 1.98 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
MLPRegressor
115 µs ± 5.04 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
FixedModel
1.13 µs ± 135 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
all_times_ms = [[i * 1000.0 / t.loops for i in t.all_runs] for t in times]

all_times_ms = [0.04186812142857l416, 0.04201067571428731,
0.05952271999998634, 0.8912386999999821, 0.10723779999999741,
                0.06644656000000201, 0.0009462411285714519]
labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']

barplot(all_times_ms, labels, ylabel="Prediction time [ms]",
filename='prediction_times')
```




```
def sum_cpu(node_res):
    cpu = sum([v['cpu'] for v in node_res])
    return cpu

def read_placement(placement, df_data, flow_dr=250):
    df_data['num_flows'].append(placement['input']['num_flows'])
    df_data['num_sources'].append(placement['input']['num_sources'])
    df_data['source_dr'].append(placement['input']['num_flows'] * flow_dr)
    df_data['num_instances'].append(placement['metrics']['num_instances'])
    df_data['max_e2e_delay'].append(placement['metrics']['max_endToEnd_delay']
)
    df_data['total_delay'].append(placement['metrics']['total_delay'])
    df_data['runtime'].append(placement['metrics']['runtime'])
    df_data['total_cpu'].append(sum_cpu(placement['placement']['alloc_node_res
']))
    return df_data

def read_results(results):
    data = {'num_sources' : [], 'num_flows': [], 'source_dr': [],
'num_instances': [],
            'max_e2e_delay': [], 'total_delay': [], 'runtime': [],
'total_cpu': []}

    # iterate through result files
    for res in glob.glob(results):
        # open and save metrics of interest
        with open(res, 'r') as f:
            placement = yaml.load(f, Loader=yaml.SafeLoader)
            data = read_placement(placement, data)

    return pd.DataFrame(data).sort_values(by=['num_flows'])
# read results
dataset = 'web_data'
sources = 'runtime'
```

```python
results = f'placement_data/{dataset}/{sources}/'

df_results = []
df_results.append(read_results(results + 'linear/*.yaml'))
df_results.append(read_results(results + 'ridge/*.yaml'))
df_results.append(read_results(results + 'svr/*.yaml'))
df_results.append(read_results(results + 'forest/*.yaml'))
df_results.append(read_results(results + 'boosting/*.yaml'))
df_results.append(read_results(results + 'mlp/*.yaml'))
df_results.append(read_results(results + 'fixed/*.yaml'))
# df_results.append(read_results(results + 'ml/*.yaml'))

labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']
placement_times = [df['runtime'] for df in df_results]

barplot(placement_times, labels)
def comparison_barplot_one(pred_times, place_times, labels, filename=None):
    sns.set(font_scale=1.1, style='white')
    assert len(pred_times) == len(place_times) == len(labels)

    # prepare data
    pred_times_mean = [np.array(t).mean() for t in pred_times]
    pred_times_std = [np.array(t).std() for t in pred_times]
    place_times_mean = [np.array(t).mean() for t in place_times]
    place_times_std = [np.array(t).std() for t in place_times]
    x = np.arange(len(labels))
    width = 0.35

    # plot: separate axes
    fig, ax1 = plt.subplots(figsize = (8, 5))
    ax1.bar(x - width/2, pred_times_mean, width, yerr=pred_times_std,
capsize=5, color='gray', label='Prediction')
    ax1.set_ylabel("Prediction time [ms]")
    ax1.set_xticks(x)
    ax1.set_xticklabels(labels)

    ax2 = ax1.twinx()
    ax2.bar(x + width/2, place_times_mean, width, yerr=place_times_std,
capsize=5, color='lightgray', label='Placement')
    ax2.set_ylabel("Placement time [s]")

    # labels
    ax1.legend()

    if filename is not None:
        fig.savefig(f'plots/{filename}.pdf', bbox_inches='tight')
```




```
from matplotlib.ticker import ScalarFormatter


def comparison_barplot_two(pred_times, place_times, labels):
    sns.set(font_scale=1.1, style='white')
    assert len(pred_times) == len(place_times) == len(labels)

    fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey='all', figsize=(8,
4))

    # prepare data
    pred_times_mean = [np.array(t).mean() for t in pred_times]
    pred_times_std = [np.array(t).std() for t in pred_times]
    # 95% CI (not bootstrapped like seaborn does):
https://en.wikipedia.org/wiki/Confidence_interval#Basic_steps
    pred_times_ci95 = [1.96 * std / np.sqrt(len(pred_times_std)) for std in
pred_times_std]

    place_times_mean = [np.array(t).mean() for t in place_times]
    place_times_std = [np.array(t).std() for t in place_times]
    place_times_ci95 = [1.96 * std / np.sqrt(len(place_times_std)) for std in
place_times_std]
    x = np.arange(len(labels))

    # plot: separate axes
    ax1.barh(x, pred_times_mean, xerr=pred_times_ci95, capsize=5,
color='gray', label='Prediction')
    ax1.set_xlabel("Prediction Time [ms]")
#     ax1.set_xscale('log')
    ax1.set_yticks(x)
    ax1.set_yticklabels(labels)
```

```
    ax1.tick_params(axis='both', direction='inout', length=5, bottom=True,
top=True)


    ax2.barh(x, place_times_mean, xerr=place_times_ci95, capsize=5,
color='lightgray', label='Placement')
    ax2.set_xlabel("Total Time [s]")
    # log scale; avoid "power 10" labels
    ax2.set_xscale('log')
    ax2.tick_params(axis='both', direction='inout', length=5, bottom=True,
top=True)
    ax2.xaxis.set_major_formatter(ScalarFormatter())

    plt.tight_layout()
    fig.savefig(f'plots/runtimes.pdf', bbox_inches='tight')
comparison_barplot_two(list(reversed(all_times_ms)),
list(reversed(placement_times)), list(reversed(labels)))
```