fase4_1

May 22, 2025

[1]:

```
import pandas as pd

df_real = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪ujindoorsubset_building1_floor2.csv")      # Dados reais de treino

df_real.describe()
```

[1]:

```
       WAP001  WAP002  WAP003  WAP004  WAP005       WAP006  WAP007  \
count  1396.0  1396.0  1396.0  1396.0  1396.0  1396.000000  1396.0
mean   -110.0  -110.0  -110.0  -110.0  -110.0  -109.584527  -110.0
std       0.0     0.0     0.0     0.0     0.0     2.912958     0.0
min    -110.0  -110.0  -110.0  -110.0  -110.0  -110.000000  -110.0
25%    -110.0  -110.0  -110.0  -110.0  -110.0  -110.000000  -110.0
50%    -110.0  -110.0  -110.0  -110.0  -110.0  -110.000000  -110.0
75%    -110.0  -110.0  -110.0  -110.0  -110.0  -110.000000  -110.0
max    -110.0  -110.0  -110.0  -110.0  -110.0   -88.000000  -110.0

            WAP008  WAP009       WAP010  …  WAP520    LONGITUDE  \
count  1396.000000  1396.0  1396.000000  …  1396.0  1396.000000
mean   -109.197708  -110.0  -109.678367  …  -110.0 -7486.581784
std       3.946364     0.0     2.181149  …     0.0    45.101037
min    -110.000000  -110.0  -110.000000  …  -110.0 -7571.093400
25%    -110.000000  -110.0  -110.000000  …  -110.0 -7520.755800
50%    -110.000000  -110.0  -110.000000  …  -110.0 -7491.030634
75%    -110.000000  -110.0  -110.000000  …  -110.0 -7443.877677
max     -80.000000  -110.0   -92.000000  …  -110.0 -7408.695251

           LATITUDE   FLOOR  BUILDINGID      SPACEID  RELATIVEPOSITION  \
count  1.396000e+03  1396.0      1396.0  1396.000000       1396.000000
mean   4.864879e+06     2.0         1.0   117.111748          1.704155
std    3.501884e+01     0.0         0.0    83.279968          0.456585
min    4.864810e+06     2.0         1.0     2.000000          1.000000
25%    4.864859e+06     2.0         1.0    17.000000          1.000000
50%    4.864873e+06     2.0         1.0   107.000000          2.000000
75%    4.864893e+06     2.0         1.0   204.000000          2.000000
max    4.864959e+06     2.0         1.0   217.000000          2.000000
```

```
            USERID      PHONEID     TIMESTAMP
count  1396.000000  1396.000000  1.396000e+03
mean      5.461318    17.108883  1.371721e+09
std       3.304272     5.297423  9.536837e+03
min       2.000000     8.000000  1.371714e+09
25%       2.000000    14.000000  1.371714e+09
50%       4.000000    18.000000  1.371715e+09
75%       9.000000    23.000000  1.371735e+09
max      10.000000    23.000000  1.371738e+09

[8 rows x 529 columns]
```

[2]:
```
df_real.columns
```

[2]:
```
Index(['WAP001', 'WAP002', 'WAP003', 'WAP004', 'WAP005', 'WAP006', 'WAP007',
       'WAP008', 'WAP009', 'WAP010',
       …
       'WAP520', 'LONGITUDE', 'LATITUDE', 'FLOOR', 'BUILDINGID', 'SPACEID',
       'RELATIVEPOSITION', 'USERID', 'PHONEID', 'TIMESTAMP'],
      dtype='object', length=529)
```

[ ]:
```
df_generated = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪df_generated_with_coords.csv") # Vetores pseudo-rotulados gerados
df_generated.describe()
```

[ ]:
```
             WAP001        WAP002        WAP003        WAP004        WAP005  \
count  40000.000000  40000.000000  40000.000000  40000.000000  40000.000000
mean     -72.304300    -66.314075    -84.710575    -55.601425    -91.486500
std        2.072235      2.581579      3.678491      3.424910      3.693394
min      -80.000000    -74.000000   -100.000000    -64.000000   -107.000000
25%      -74.000000    -68.000000    -87.000000    -58.000000    -94.000000
50%      -72.000000    -66.000000    -84.000000    -56.000000    -91.000000
75%      -71.000000    -65.000000    -82.000000    -53.000000    -89.000000
max      -65.000000    -57.000000    -76.000000    -40.000000    -80.000000

           WAP006        WAP007        WAP008        WAP009        WAP010  \
count  40000.0000  40000.000000  40000.000000  40000.000000  40000.000000
mean     -56.7046    -86.928525    -59.681625    -80.286775    -92.850525
std        3.7596      4.390803      2.787877      3.347576      3.430230
min      -66.0000   -107.000000    -70.000000    -95.000000   -110.000000
25%      -60.0000    -90.000000    -62.000000    -82.000000    -95.000000
50%      -57.0000    -87.000000    -60.000000    -80.000000    -92.000000
75%      -54.0000    -84.000000    -58.000000    -78.000000    -90.000000
max      -42.0000    -75.000000    -49.000000    -73.000000    -85.000000

          LONGITUDE      LATITUDE
count  40000.000000  40000.000000
```

```
mean       2.852250    12.679366
std        1.591303     1.664991
min       -0.164581     8.877346
25%        1.526815    11.362357
50%        2.732364    12.441347
75%        3.963216    13.768739
max        8.701089    19.392925
```

[4]:
```
df_generated.columns
```

[4]:
```
Index(['WAP001', 'WAP002', 'WAP003', 'WAP004', 'WAP005', 'WAP006', 'WAP007',
       'WAP008', 'WAP009', 'WAP010', 'LONGITUDE', 'LATITUDE'],
      dtype='object')
```

[7]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from IPython.display import display

# 1. Carregar datasets
df_real = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪ujindoorsubset_building1_floor2.csv")
df_generated = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪df_generated_with_coords.csv")
df_synthetic = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪df_selected_synthetic.csv")
df_test = pd.read_csv("/home/darkcover/Documentos/Gan/Data/df_test.csv")
df_simulated = pd.read_csv("/home/darkcover/Documentos/Gan/Data/df_simulated.
 ↪csv")
```

[9]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

# --------------------------------------------------
# 1) Experimento de Simulação (Tabela 2 + Figura 6)
# --------------------------------------------------

# 1.1) Carregar dados de simulação gerados em Fase 1
df_sim = df_simulated  # 1000×(10 WAP + X,Y)
df_tst = df_test         # 800×(10 WAP + X,Y)

# 1.2) Preparar X/y
wap_sim1 = [c for c in df_sim.columns if c.startswith("WAP")]
wap_tst1 = [c for c in df_tst.columns if c.startswith("WAP")]
```

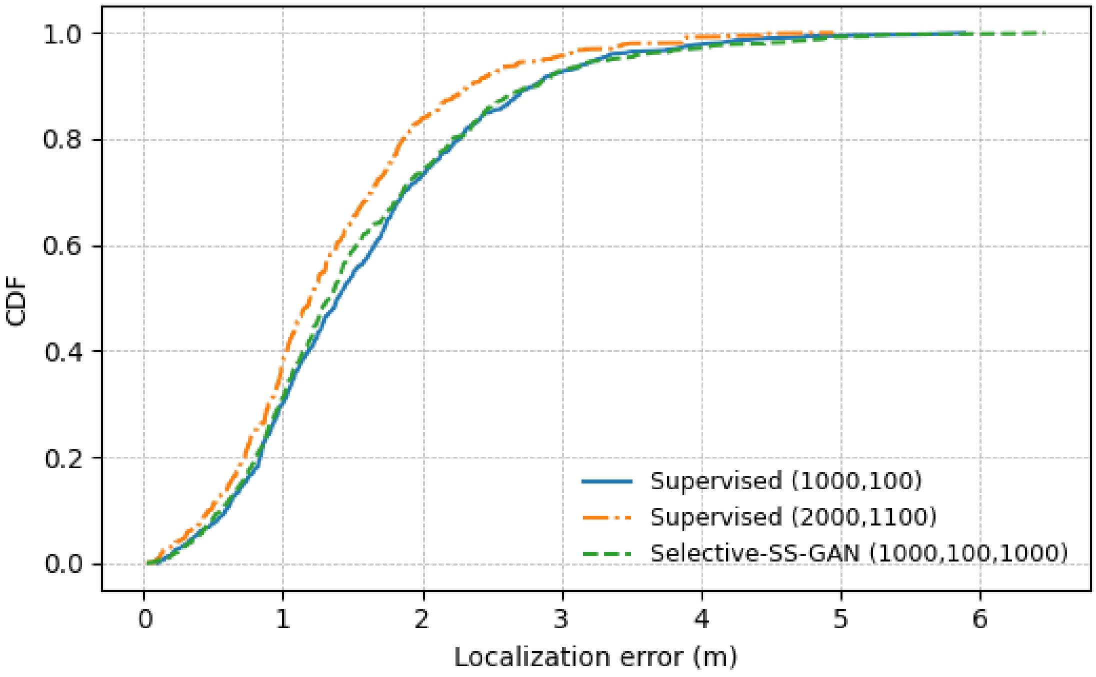
```
X_sim = df_sim[wap_sim1].values.astype(np.float32)
y_sim = df_sim[["X","Y"]].values.astype(np.float32)
X_tst = df_tst[wap_tst1].values.astype(np.float32)
y_tst = df_tst[["X","Y"]].values.astype(np.float32)

# 1.3) Funções auxiliares
def build_model(input_dim, lr=0.01):
    m = tf.keras.Sequential([
        tf.keras.layers.Dense(128,
                              activation="relu",
                              input_shape=(input_dim,)),   # tupla!
        tf.keras.layers.Dense(64,  activation="relu"),
        tf.keras.layers.Dense(2)
    ])
    m.compile(optimizer=tf.keras.optimizers.Adam(lr), loss="mse")
    return m

def eval_errs(X_tr, y_tr, X_ev, y_ev, epochs, bs, lr):
    m = build_model(X_tr.shape[1], lr)
    m.fit(X_tr, y_tr, epochs=epochs, batch_size=bs, verbose=0)
    y_pred = m.predict(X_ev, verbose=0)
    return np.linalg.norm(y_pred - y_ev, axis=1)

# 1.4) Experimentos de simulação
errs_sup100   = eval_errs(X_sim, y_sim, X_tst, y_tst, epochs=250, bs=100, lr=0.
 ↪01)
# duplicar para (2000,1100)
X2 = np.vstack([X_sim, X_sim]); y2 = np.vstack([y_sim, y_sim])
# gerar test-set de 1100 amostras aleatórias
idx1100   = np.random.choice(len(X_tst), 1100, replace=True)
X_tst1100 = X_tst[idx1100]; y_tst1100 = y_tst[idx1100]
errs_sup2000 = eval_errs(X2, y2, X_tst1100, y_tst1100, epochs=250, bs=100, lr=0.
 ↪01)

errs_sel = {}
for ms in [100, 500, 1000]:
    key = f"Selective-SS-GAN(1000,100,{ms})"
    # carregar apenas os primeiros ms sintéticos
    df_gen = df_generated # 40000×10
    wap_gen = [c for c in df_gen.columns if c.startswith("WAP")]
    Xg = df_gen[wap_gen].values.astype(np.float32)[:ms]
    yg = df_gen[["X","Y"]].values.astype(np.float32)[:ms]
    X_mix = np.vstack([X_sim, Xg]); y_mix = np.vstack([y_sim, yg])
    errs_sel[ms] = eval_errs(X_mix, y_mix, X_tst, y_tst, epochs=250, bs=100,␣
 ↪lr=0.01)

# 1.5) Montar Tabela 2
```

```
rows = [
    ("Supervised (1000,100)",   errs_sup100),
    ("Supervised (2000,1100)",  errs_sup2000),
]
rows += [(f"Selective-SS-GAN (1000,100,{ms})", errs_sel[ms]) for ms in␣
  ↪[100,500,1000]]

df2 = pd.DataFrame([{
    "Método": name,
    "Erro médio (m)": e.mean(),
    "Erro mínimo (m)": e.min(),
    "Erro máximo (m)": e.max()
} for name,e in rows])
styled2 = (df2.style.hide(axis="index")
              .set_caption("Tabela 2. Simulação - performance de localização")
              .format({"Erro médio (m)":"{:.3f}", "Erro mínimo (m)":"{:.3f}",␣
  ↪"Erro máximo (m)":"{:.3f}"})
              .set_table_styles([
                  {"selector":"caption","props":
  ↪[("caption-side","bottom"),("font-weight","bold"),("text-align","center")]},
                  {"selector":"th","props":
  ↪[("font-weight","bold"),("text-align","center")]},
                  {"selector":"td","props":[("text-align","center")]}]))
display(styled2)

# 1.6) Plotar Figura 6 (CDF de simulação)
def plot_cdf(errs, style, label):
    s = np.sort(errs)
    c = np.arange(len(s)) / float(len(s))
    plt.plot(s, c, linestyle=style, linewidth=1.5, label=label)

plt.figure(figsize=(6,4), dpi=100)
plot_cdf(errs_sup100,   "-",  "Supervised (1000,100)")
plot_cdf(errs_sup2000, "-.",  "Supervised (2000,1100)")
plot_cdf(errs_sel[1000], "--","Selective-SS-GAN (1000,100,1000)")
plt.xlabel("Localization error (m)"); plt.ylabel("CDF")
plt.title("Fig. 6. Simulação - comparação de desempenho", fontsize=11,␣
  ↪fontweight="bold")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.legend(frameon=False, loc="lower right", fontsize=9)
plt.tight_layout()
plt.show()

# --------------------------------------------------
# 2) Experimento no UJIndoorLoc real (Tabela 3)
# --------------------------------------------------
# 2.1) Carregar e processar UJIndoorLoc
df_full = df_real              # :contentReference[oaicite:1]{index=1}
# missing=100 → -110
wap_full = [c for c in df_full.columns if c.startswith("WAP")]
df_full[wap_full] = df_full[wap_full].replace(100, -110)
# filtrar Building 1, Floor 2
df_real = df_full[(df_full.BUILDINGID==1) & (df_full.FLOOR==2)].copy()
# colunas de coord
X_real = df_real[wap_full].values.astype(np.float32)
y_real = df_real[["LONGITUDE","LATITUDE"]].values.astype(np.float32)

# 2.2) Carregar pseudo-rotulados (Fase 3)
df_sel = df_synthetic # 1000×(10 WAP + X,Y)
# filtrar Building 1, Floor 2
wap_sel = [c for c in df_sel.columns if c.startswith("WAP")]
Xg = df_sel[wap_sel].values.astype(np.float32)
yg = df_sel[["X","Y"]].values.astype(np.float32)

# 2.3) Avaliar real em supervised (1000 reais) e com seleção inteligente
# (usar y_real para test real)
# separar 1000 primeiros reais para treino
Xr1000, yr1000 = X_real[:1000], y_real[:1000]
# validar em todo o conjunto real restante
Xr_val, yr_val = X_real[1000:], y_real[1000:]
errs_r1000 = eval_errs(Xr1000, yr1000, Xr_val, yr_val, epochs=250, bs=100, lr=0.
 ↪01)
# misturar 1000 reais + 1000 sintéticos selecionados
X_mix = np.vstack([Xr1000, Xg[:1000]]); y_mix = np.vstack([yr1000, yg[:1000]])
errs_mix = eval_errs(X_mix, y_mix, Xr_val, yr_val, epochs=250, bs=100, lr=0.01)

# 2.4) Montar Tabela 3
df3 = pd.DataFrame([
    {"Método":"Supervised (1000)",    "Erro médio (m)":errs_r1000.mean(), "Erro␣
 ↪mínimo (m)":errs_r1000.min(), "Erro máximo (m)":errs_r1000.max()},
    {"Método":"Selective-SS-GAN (1000)", "Erro médio (m)":errs_mix.mean(), ␣
 ↪"Erro mínimo (m)":errs_mix.min(),     "Erro máximo (m)":errs_mix.max()},
])
styled3 = (df3.style.hide(axis="index")
               .set_caption("Tabela 3. UJIndoorLoc - 1000 reais vs. +1000␣
 ↪sintéticos")
               .format({"Erro médio (m)":"{:.3f}", "Erro mínimo (m)":"{:.3f}",␣
 ↪"Erro máximo (m)":"{:.3f}"})
               .set_table_styles([
                   {"selector":"caption","props":
 ↪[("caption-side","bottom"),("font-weight","bold"),("text-align","center")]},
                   {"selector":"th","props":
↪[("font-weight","bold"),("text-align","center")]},
                   {"selector":"td","props":[("text-align","center")]}]))
display(styled3)
```

```
/home/darkcover/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-
py3.12/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/home/darkcover/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-
py3.12/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/home/darkcover/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-
py3.12/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/home/darkcover/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-
py3.12/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
<pandas.io.formats.style.Styler at 0x736471c81490>
```

Fig. 6. Simulação — comparação de desempenho



```
/home/darkcover/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-
py3.12/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[9], line 129
    127 errs_r1000 = eval_errs(Xr1000, yr1000, Xr_val, yr_val, epochs=250,␣
  ↪bs=100, lr=0.01)
    128 # misturar 1000 reais + 1000 sintéticos selecionados
--> 129 X_mix = np.vstack([Xr1000, Xg[:1000]]); y_mix = np.vstack([yr1000, yg[:
  ↪1000]])
    130 errs_mix = eval_errs(X_mix, y_mix, Xr_val, yr_val, epochs=250, bs=100,␣
  ↪lr=0.01)
    132 # 2.4) Montar Tabela 3

File ~/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-py3.12/lib/python3.12/
  ↪site-packages/numpy/_core/shape_base.py:291, in vstack(tup, dtype, casting)
    289 if not isinstance(arrs, tuple):
    290     arrs = (arrs,)
--> 291 return _nx.concatenate(arrs, 0, dtype=dtype, casting=casting)
```

```
ValueError: all the input array dimensions except for the concatenation axis␣
  ↪must match exactly, but along dimension 1, the array at index 0 has size 520␣
  ↪and the array at index 1 has size 10
```