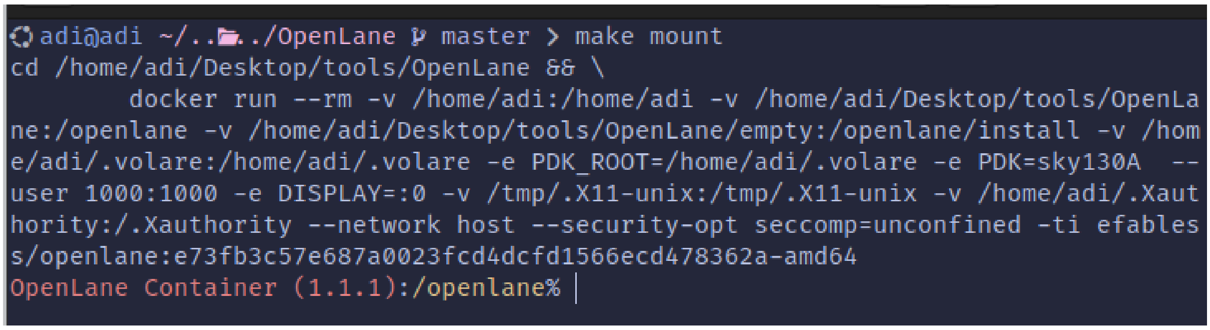
ML - Based Floor Plan Optimiser Using OpenLane:
TEAM:
Adithya V
Tarun Kumar
Livingstone

Requirements:
Software : OpenLane (v 1.1.1) , Klayout
Language : Python (version 3)
Technology File : Skywater 130nm PDK

OpenLane Execution:

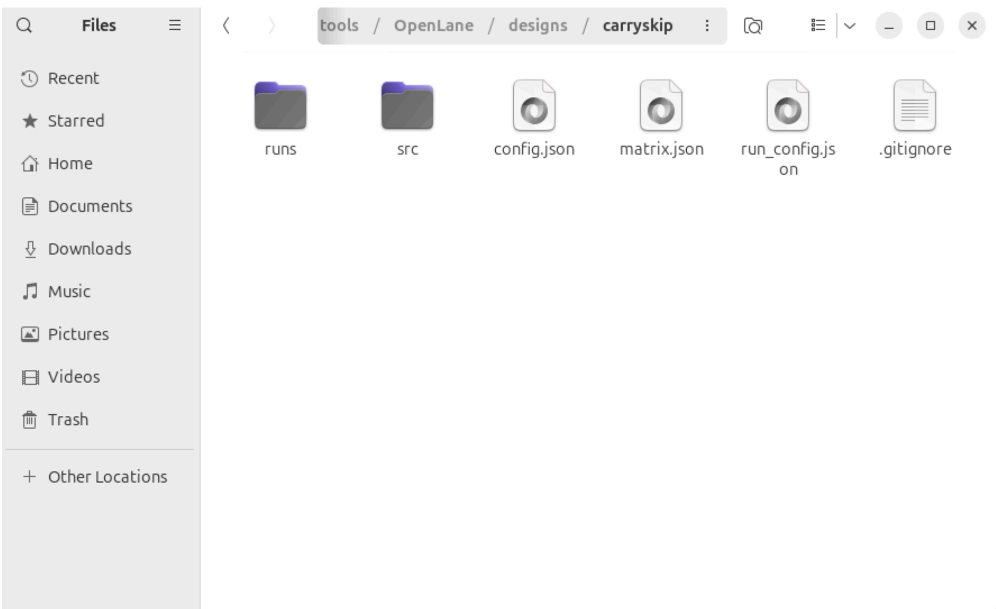


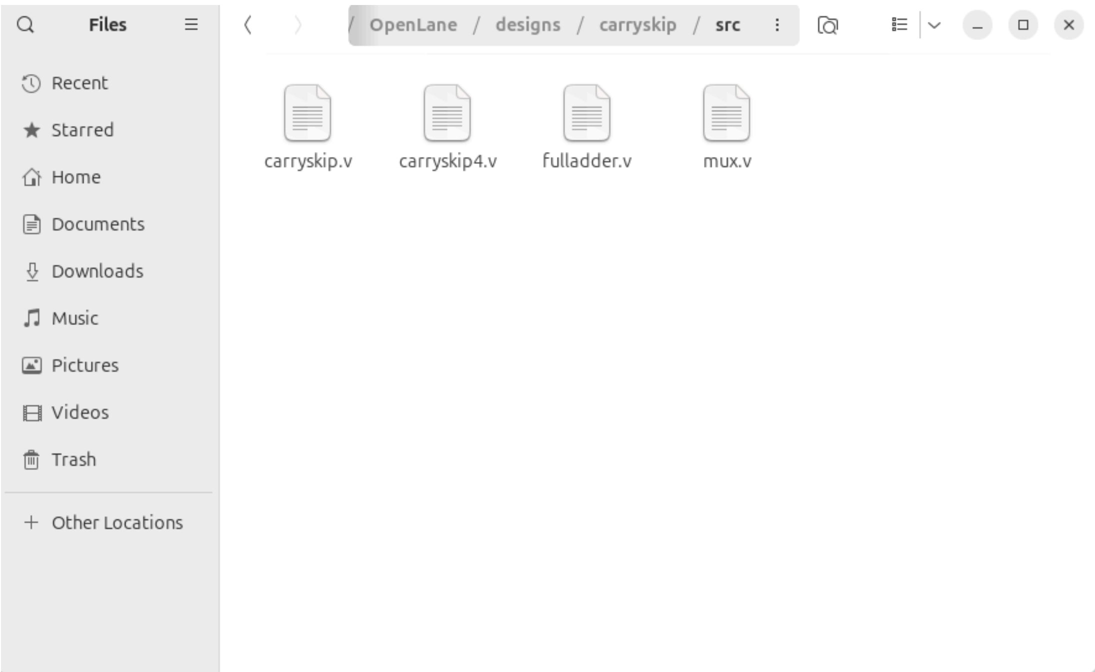
```
adi@adi ~/../OpenLane  master > make mount
cd /home/adi/Desktop/tools/OpenLane && \
        docker run --rm -v /home/adi:/home/adi -v /home/adi/Desktop/tools/OpenLane:/openlane -v /home/adi/Desktop/tools/OpenLane/empty:/openlane/install -v /home/adi/.volare:/home/adi/.volare -e PDK_ROOT=/home/adi/.volare -e PDK=sky130A  --user 1000:1000 -e DISPLAY=:0 -v /tmp/.X11-unix:/tmp/.X11-unix -v /home/adi/.Xauthority:/.Xauthority --network host --security-opt seccomp=unconfined -ti efabless/openlane:e73fb3c57e687a0023fcd4dcfd1566ecd478362a-amd64
OpenLane Container (1.1.1):/openlane%
```

The make mount command in OpenLane is used to launch OpenLane inside a Docker container with your local project files and PDKs properly mounted.

We can either use **run_designs.py** or **flow.tcl** to run our own designs in OpenLane

Inside the OpenLane Design Folder we create our own folder naming “Carryskip” and place all the verilog files required in src folder

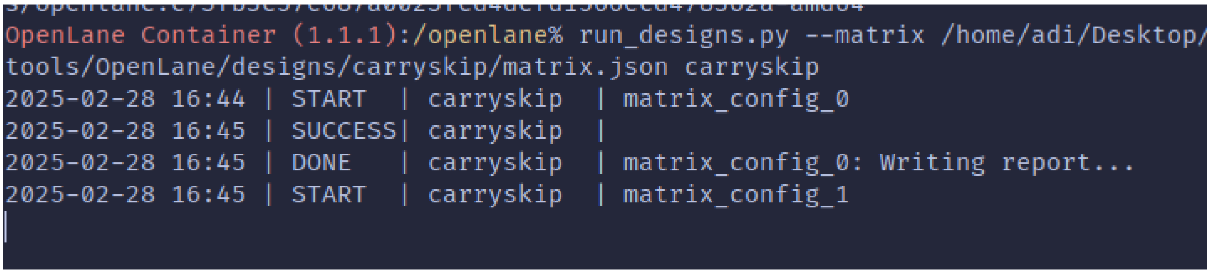




Matrix.json is the file which is used while running Openlane to obtain the Dataset for the ML code to train with

```json
{
  "FP_CORE_UTIL": [35,36,37,38,39,40,41,42,43,44,45],
  "DIE_AREA": [500,600,750],
  "CORE_AREA": [500,600,700],
  "PL_TARGET_DENSITY": [0.5,0.6,0.65,0.7,0.75]
}
```

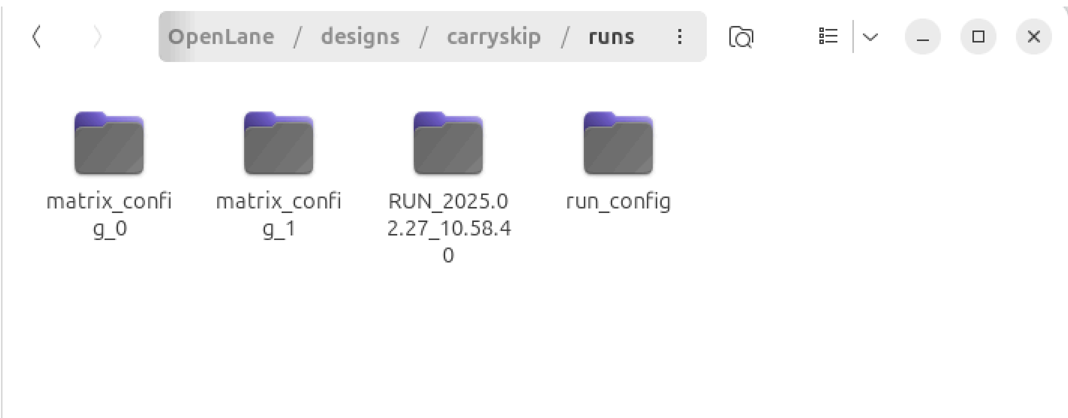
This is format of Matrix.json this will create n number of different configuration of the above given values and produce the config.json file individually running openlane for each of the above



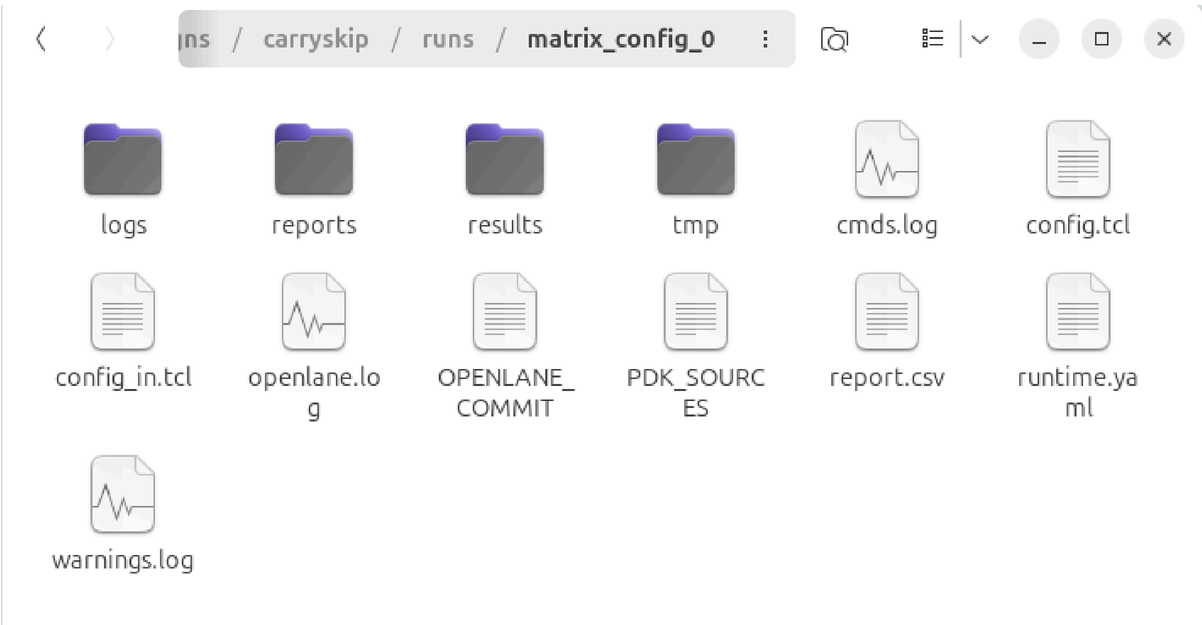
This is command to run the multiple runs in a series order continuously

These are multiple .json files created by the program to run individually

Results:



In the runs Folder we can find all the runs performed by the OpenLane in different configurations , if a run is successful we can find all the results of that design in report.csv and 22-global.log and also the .gds file to view the final layout of the design in that particular configuration



The report.csv have the Wirelength , DIE area , Power values so to extract is from the each of the run which are in different folders we use a python code to

	0	1	2	3	4	5	6	7	8
0	design	design_name	config	flow_status	total_runtime	routed_runtime	DIEAREA_mm^2	CellPer_mm^2	OpenDP
1	carryskip	carryskip	matrix_config_0	flow completed	0h0m44s0ms	0h0m31s0ms	0.0025569929	30113.49777310684	-1

extract it from multiple files to a single .csv file

Conjustion data is found in the 22-global.log file in ../log/routing/ to the extract the data from it we need to use reg ex in the python Script

```python
import re
import pandas as pd
import os

# Function to extract values from config.tcl
def extract_config_params(config_path):
    params = {
        "FP_CORE_UTIL": None,
        "DIE_AREA": None,
        "CORE_AREA": None,
        "PL_TARGET_DENSITY": None
    }

    if os.path.exists(config_path):
        with open(config_path, "r") as file:
            for line in file:
                for key in params.keys():
                    match = re.search(rf"set ::env\({key}\) (.+)", line)
                    if match:
                        try:
                            params[key] = float(match.group(1).strip().replace('"', ''))  # Convert to float, remove quotes
                        except ValueError:
                            params[key] = match.group(1).strip().replace('"', '')  # Remove quotes if not a number

    return params

# List to store dataframes
dataframes = []

# Iterate over matrix_config folders
for i in range(570):  # Assuming runs 0 to 82
    base_path = f"/home/adi/Desktop/tools/OpenLane/designs/carryskip/runs/matrix_config_{i}"
    report_path = f"{base_path}/report.csv"
    config_path = f"{base_path}/config_in.tcl"

    try:
        # Read report.csv
        df = pd.read_csv(report_path)

        # Extract parameters from config.tcl
        config_params = extract_config_params(config_path)

        # Add extracted parameters to the DataFrame
        for key, value in config_params.items():
            df[key] = value

        # Append to list
        dataframes.append(df)
```

This code is to extract from the reports.csv and merge all the 570 runs

```python

    except FileNotFoundError:
        print(f"Warning: {report_path} not found. Skipping...")

# Merge all DataFrames
if dataframes:
    merged_df = pd.concat(dataframes, ignore_index=True)
    merged_df.to_csv("merged_output.csv", index=False)
    print(f"Successfully merged {len(dataframes)} CSV files into 'merged_output.csv'.")
else:
    print("No CSV files found. Nothing to merge.")
```

```python
import re
import pandas as pd

# Path to the existing merged CSV file
csv_path = "/home/adi/Documents/Scripts/Python/merged_output.csv"

# Read the existing CSV file
df = pd.read_csv(csv_path)

# List to store congestion values
congestion_values = []

# Iterate through the log files to extract congestion data
for i in range(570):
    log_path = f"/home/adi/Desktop/tools/OpenLane/designs/carryskip/runs/matrix_config_{i}/logs/routing/22-global.log
    try:
        with open(log_path, "r") as f:
            log_content = f.read()

        # Adjusted regex pattern with flexible spaces
        pattern = r"Total\s+(\d+)\s+(\d+)\s+([\d.]+)%\s+(\d+)\s*/\s*(\d+)\s*/\s*(\d+)"
        match = re.search(pattern, log_content)

        if match:
            total_congestion = float(match.group(3))  # Extract total congestion percentage
        else:
            print(f"No match found in: {log_path}")
            total_congestion = None  # Assign None if no match found
    except FileNotFoundError:
        print(f"File not found: {log_path}")
        total_congestion = None

    congestion_values.append(total_congestion)

# Ensure the length matches the existing dataframe
while len(congestion_values) < len(df):
    congestion_values.append(None)

# Add congestion data to the dataframe
df["Congestion"] = congestion_values[:len(df)]

# Save the updated dataframe
output_path = "/home/adi/Documents/Scripts/Python/merged_output_updated1.csv"
df.to_csv(output_path, index=False)

print(f"Updated CSV saved to {output_path}")
```

This code is extract conjunction data and add as last column in the final added .csv file

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	design	design_name	config	flow_status	total_runtime	routed_runtime	DIEAREA_mm^2	CellPer_mm^2	OpenDP_Util	Final_Util	Peak_Memory_Usage_MB	synth_cell_count	tritonRoute_violations	Short_viola
1	carryskip	carryskip	matrix_config_0	flow completed	0h0m43s0ms	0h0m30s0ms	0.0028453025	27062.148927925944	-1	32.6923	483.02	51	0	0
2	carryskip	carryskip	matrix_config_1	flow completed	0h0m45s0ms	0h0m31s0ms	0.002780800425	27689.86918577589	-1	33.1061	490.48	51	0	0
3	carryskip	carryskip	matrix_config_2	flow completed	0h0m44s0ms	0h0m30s0ms	0.0027201752999999	28306.99918494224	-1	33.5306	491.03	51	0	0
4	carryskip	carryskip	matrix_config_3	flow completed	0h0m45s0ms	0h0m31s0ms	0.0026628048	28916.877421882367	-1	34.165800000000004	489.91	51	0	0
5	carryskip	carryskip	matrix_config_4	flow completed	0h0m44s0ms	0h0m30s0ms	0.002608606425	29517.6762818868	-1	37.8889	487.64	51	0	0
6	carryskip	carryskip	matrix_config_5	flow completed	0h0m48s0ms	0h0m34s0ms	0.0025569929	30113.49777310684	-1	38.4009	479.48	51	0	0
7	carryskip	carryskip	matrix_config_6	flow completed	0h0m48s0ms	0h0m33s0ms	0.0025084073	30696.769220851813	-1	38.9269	486.45	51	0	0
8	carryskip	carryskip	matrix_config_7	flow completed	0h0m47s0ms	0h0m32s0ms	0.0028453025	27062.148927925944	-1	32.6923	495.77	51	0	0
9	carryskip	carryskip	matrix_config_8	flow completed	0h0m47s0ms	0h0m33s0ms	0.002780800425	27689.86918577589	-1	33.1061	479.29	51	0	0
10	carryskip	carryskip	matrix_config_9	flow completed	0h0m48s0ms	0h0m33s0ms	0.0027201752999999	28306.99918494224	-1	33.5306	496.87	51	0	0
11	carryskip	carryskip	matrix_config_10	flow completed	0h0m47s0ms	0h0m33s0ms	0.0026628048	28916.877421882367	-1	34.165800000000004	480.0	51	0	0
12	carryskip	carryskip	matrix_config_11	flow completed	0h0m47s0ms	0h0m32s0ms	0.002608606425	29517.6762818868	-1	37.8889	478.56	51	0	0
13	carryskip	carryskip	matrix_config_12	flow completed	0h0m46s0ms	0h0m32s0ms	0.0025569929	30113.49777310684	-1	38.4009	479.45	51	0	0
14	carryskip	carryskip	matrix_config_13	flow completed	0h0m47s0ms	0h0m33s0ms	0.0025084073	30696.769220851813	-1	38.9269	478.54	51	0	0
15	carryskip	carryskip	matrix_config_14	flow completed	0h0m47s0ms	0h0m33s0ms	0.0028453025	27062.148927925944	-1	32.6923	491.52	51	0	0
16	carryskip	carryskip	matrix_config_15	flow completed	0h0m46s0ms	0h0m32s0ms	0.002780800425	27689.86918577589	-1	33.1061	479.33	51	0	0
17	carryskip	carryskip	matrix_config_16	flow completed	0h0m45s0ms	0h0m32s0ms	0.0027201752999999	28306.99918494224	-1	33.5306	481.62	51	0	0
18	carryskip	carryskip	matrix_config_17	flow completed	0h0m45s0ms	0h0m32s0ms	0.0026628048	28916.877421882367	-1	34.165800000000004	487.69	51	0	0
19	carryskip	carryskip	matrix_config_18	flow completed	0h0m46s0ms	0h0m32s0ms	0.002608606425	29517.6762818868	-1	37.8889	481.04	51	0	0
20	carryskip	carryskip	matrix_config_19	flow completed	0h0m46s0ms	0h0m32s0ms	0.0025569929	30113.49777310684	-1	38.4009	486.17	51	0	0
21	carryskip	carryskip	matrix_config_20	flow completed	0h0m46s0ms	0h0m32s0ms	0.0025084073	30696.769220851813	-1	38.9269	487.43	51	0	0
22	carryskip	carryskip	matrix_config_21	flow completed	0h0m46s0ms	0h0m32s0ms	0.0028453025	27062.148927925944	-1	32.6923	486.75	51	0	0
23	carryskip	carryskip	matrix_config_22	flow completed	0h0m46s0ms	0h0m32s0ms	0.002780800425	27689.86918577589	-1	33.1061	487.84	51	0	0
24	carryskip	carryskip	matrix_config_23	flow completed	0h0m45s0ms	0h0m32s0ms	0.0027201752999999	28306.99918494224	-1	33.5306	487.88	51	0	0
25	carryskip	carryskip	matrix_config_24	flow completed	0h0m46s0ms	0h0m32s0ms	0.0026628048	28916.877421882367	-1	34.165800000000004	487.04	51	0	0
26	carryskip	carryskip	matrix_config_25	flow completed	0h0m45s0ms	0h0m31s0ms	0.002608606425	29517.6762818868	-1	37.8889	487.84	51	0	0
27	carryskip	carryskip	matrix_config_26	flow completed	0h0m45s0ms	0h0m31s0ms	0.0025569929	30113.49777310684	-1	38.4009	478.3	51	0	0
28	carryskip	carryskip	matrix_config_27	flow completed	0h0m45s0ms	0h0m31s0ms	0.0025084073	30696.769220851813	-1	38.9269	486.73	51	0	0
29	carryskip	carryskip	matrix_config_28	flow completed	0h0m45s0ms	0h0m31s0ms	0.0028453025	27062.148927925944	-1	32.6923	489.27	51	0	0
30	carryskip	carryskip	matrix_config_29	flow completed	0h0m47s0ms	0h0m32s0ms	0.002780800425	27689.86918577589	-1	33.1061	487.35	51	0	0
31	carryskip	carryskip	matrix_config_30	flow completed	0h0m46s0ms	0h0m32s0ms	0.0027201752999999	28306.99918494224	-1	33.5306	489.09	51	0	0
32	carryskip	carryskip	matrix_config_31	flow completed	0h0m45s0ms	0h0m31s0ms	0.0026628048	28916.877421882367	-1	34.165800000000004	487.88	51	0	0
33	carryskip	carryskip	matrix_config_32	flow completed	0h0m44s0ms	0h0m31s0ms	0.002608606425	29517.6762818868	-1	37.8889	486.19	51	0	0
34	carryskip	carryskip	matrix_config_33	flow completed	0h0m44s0ms	0h0m31s0ms	0.0025569929	30113.49777310684	-1	38.4009	488.3	51	0	0
35	carryskip	carryskip	matrix_config_34	flow completed	0h0m44s0ms	0h0m31s0ms	0.0025084073	30696.769220851813	-1	38.9269	479.55	51	0	0
36	carryskip	carryskip	matrix_config_35	flow completed	0h0m45s0ms	0h0m32s0ms	0.0028453025	27062.148927925944	-1	32.6923	495.32	51	0	0

Now to discard all the unwanted parameters and add the input parameters in the file for the training of the ML Model

```python
import pandas as pd
import os
import glob

# 📍 Update this path to your folder containing CSV files
folder_path = "/home/adi/Documents/Scripts/csv"  # Change this to your actual path

# 🔍 Find all CSV files in the folder
csv_files = glob.glob(os.path.join(folder_path, "*.csv"))

# 🛑 Check if CSV files exist
if not csv_files:
    print("❌ No CSV files found! Check your folder path.")
    exit()

# 📚 Define the required columns
selected_columns = [
    "DIEAREA_mm^2", "CORE_AREA","DIE_AREA", "FP_CORE_UTIL", "PL_TARGET_DENSITY",
    "power_typical_switching_uW", "Congestion", "wire_length"
]

# 🏗 Read and merge only the selected columns
df_list = []
for file in csv_files:
    try:
        df = pd.read_csv(file, usecols=selected_columns)
        df_list.append(df)
        print(f"✅ Successfully loaded: {file}")
    except Exception as e:
        print(f"⚠️ Error loading {file}: {e}")

# 🔄 Combine all dataframes
combined_df = pd.concat(df_list, ignore_index=True)

# 💾 Save the merged CSV file
output_file = os.path.join(folder_path, "merged.csv")
combined_df.to_csv(output_file, index=False)

print(f"\n🎉 Merged CSV saved as: {output_file}")
```

Now that we have all the files in a single file we can Use

Explanation of the Code

This script trains a **Machine Learning-based floorplan optimizer** for OpenLane using **Random Forest Regression** and **Optuna-based hyperparameter optimization**.

---

1. Overview of What the Code Does

1. **Loads floorplanning-related data** from a CSV file.

2. **Trains a multi-output regression model** to predict design metrics like wirelength, power, congestion, etc.
3. **Optimizes input parameters** (e.g., `DIE_AREA`, `CORE_AREA`, `FP_CORE_UTIL`, `PL_TARGET_DENSITY`) to minimize the predicted floorplan metrics using **Optuna**.
4. **Visualizes the optimization results** and **saves the trained model** for future use.

---

2. Step-by-Step Breakdown

Step 1: Load and Prepare Data (`load_data`)

- Reads a CSV file (`merged.csv`) containing floorplanning metrics.
- Extracts **input features** (design parameters) and **output features** (performance metrics).
- **Inputs (X)**:
  - `DIE_AREA` (Chip Die Area)
  - `CORE_AREA` (Core Area)
  - `FP_CORE_UTIL` (Floorplan Core Utilization)
  - `PL_TARGET_DENSITY` (Placement Density)
- **Outputs (y)**:
  - `wire_length`
  - `DIEAREA_mm^2`
  - `power`
  - `Congestion`

Step 2: Train Prediction Model (`train_prediction_model`)

- **Preprocessing:**
  - Splits the dataset into **80% training** and **20% testing**.
  - Uses `StandardScaler` to normalize both inputs and outputs.
- **Model Choice:**

    - Uses **Random Forest Regressor** inside **MultiOutputRegressor** for multi-output regression.
    - This means one Random Forest model is trained for each output feature.
- **Model Training:**
    - Fits the model on **scaled** training data.
    - Evaluates using **Mean Squared Error (MSE)** and **R^2 Score**.

Step 3: Define Optimization Objective (`create_objective_function`)

- Creates a function that:
    - Takes a **set of design parameters**.
    - Uses the trained model to **predict output metrics**.
    - Computes a **weighted cost** (sum of wirelength, area, power, congestion).
    - Returns the **cost to minimize**.
- **Why use weights?**
    - To control the importance of each metric in optimization.
    - Example: If reducing power is most important, assign it a higher weight.

Step 4: Optimize Parameters Using Optuna (`optimize_parameters_optuna`)

- **Uses Optuna to find the best floorplan parameters** that minimize cost.
- **How it works:**
    1. Suggests random values for **DIE_AREA, CORE_AREA, FP_CORE_UTIL, PL_TARGET_DENSITY** within their data range.
    2. Predicts the corresponding **wirelength, area, power, and congestion** using the trained model.
    3. Computes the weighted cost.
    4. Optuna **iterates over 200 trials** to find the best combination of parameters.

- **Best parameters** are stored and their corresponding output values are predicted.

Step 5: Visualize Results (`visualize_results`)

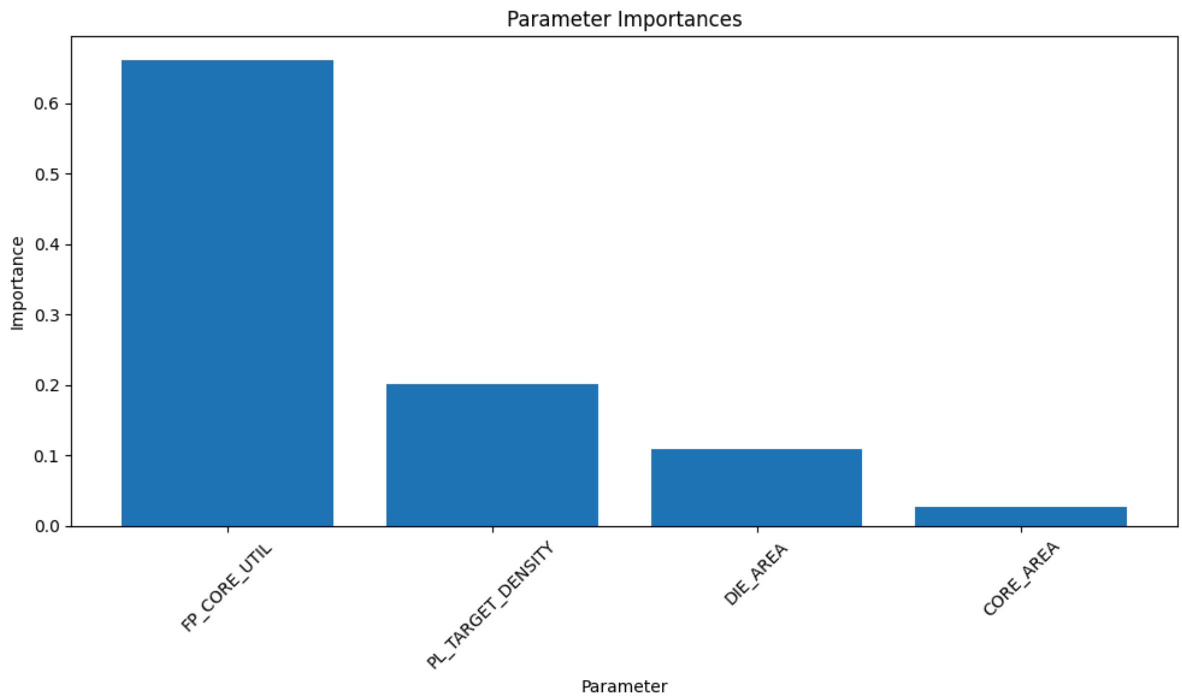
- **Saves two plots:**
  1. **Optimization history** (tracks how the cost improves over trials).
  2. **Feature importance** (shows which input parameters impact results most).

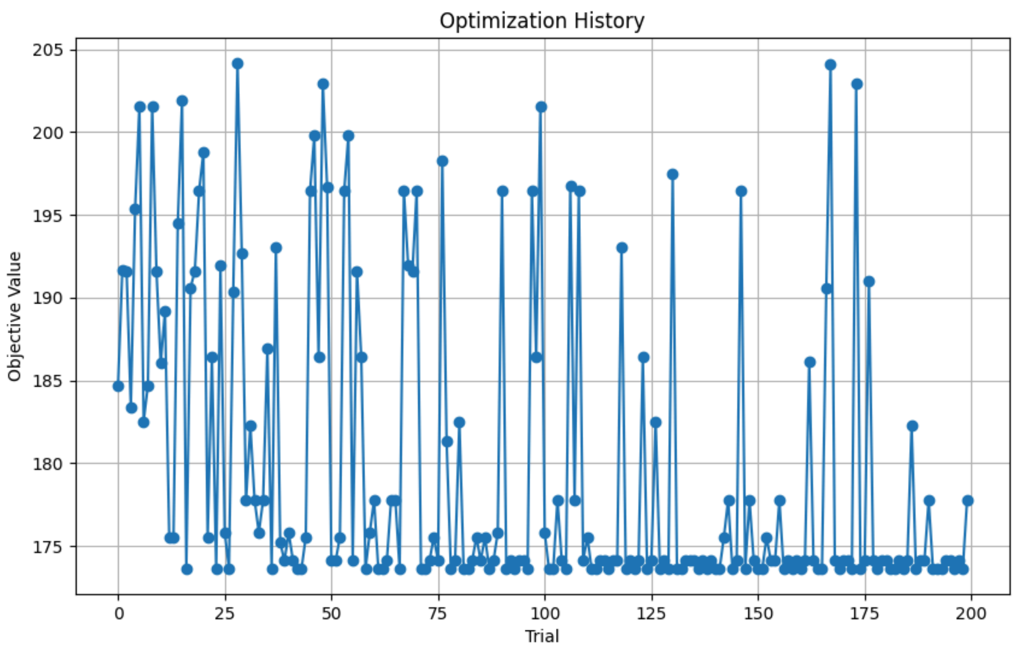
Step 6: Save Model and Results (`main`)

1. Loads and preprocesses data.
2. Trains the **Random Forest-based multi-output regression model**.
3. Runs **Optuna-based optimization**.
4. Saves the trained model as `floorplan_optimizer_model.pkl` using `joblib`

After running this in a virtual Environment we get the values

```
Optimal Input Parameters:
DIE_AREA: 657.3198181828924
CORE_AREA: 591.2152672198778
FP_CORE_UTIL: 44.88092141576208
PL_TARGET_DENSITY: 0.6766801503823773

Predicted Outputs:
wire_length: 686.0
DIEAREA_mm^2: 0.0021596388000000005
power: 2.33e-05
Congestion: 8.47

Parameter Importances:
FP_CORE_UTIL: 0.661534008207499
PL_TARGET_DENSITY: 0.20149127781916856
DIE_AREA: 0.10899578534754123
CORE_AREA: 0.02797892862579122
```


Parameter Importances
Importance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
FP_CORE_UTIL
PL_TARGET_DENSITY
DIE_AREA
CORE_AREA
Parameter


Optimization History
Objective Value
175
180
185
190
195
200
205
0
25
50
75
100
125
150
175
200
Trial

Results and Discussion:

Initial example Configuration

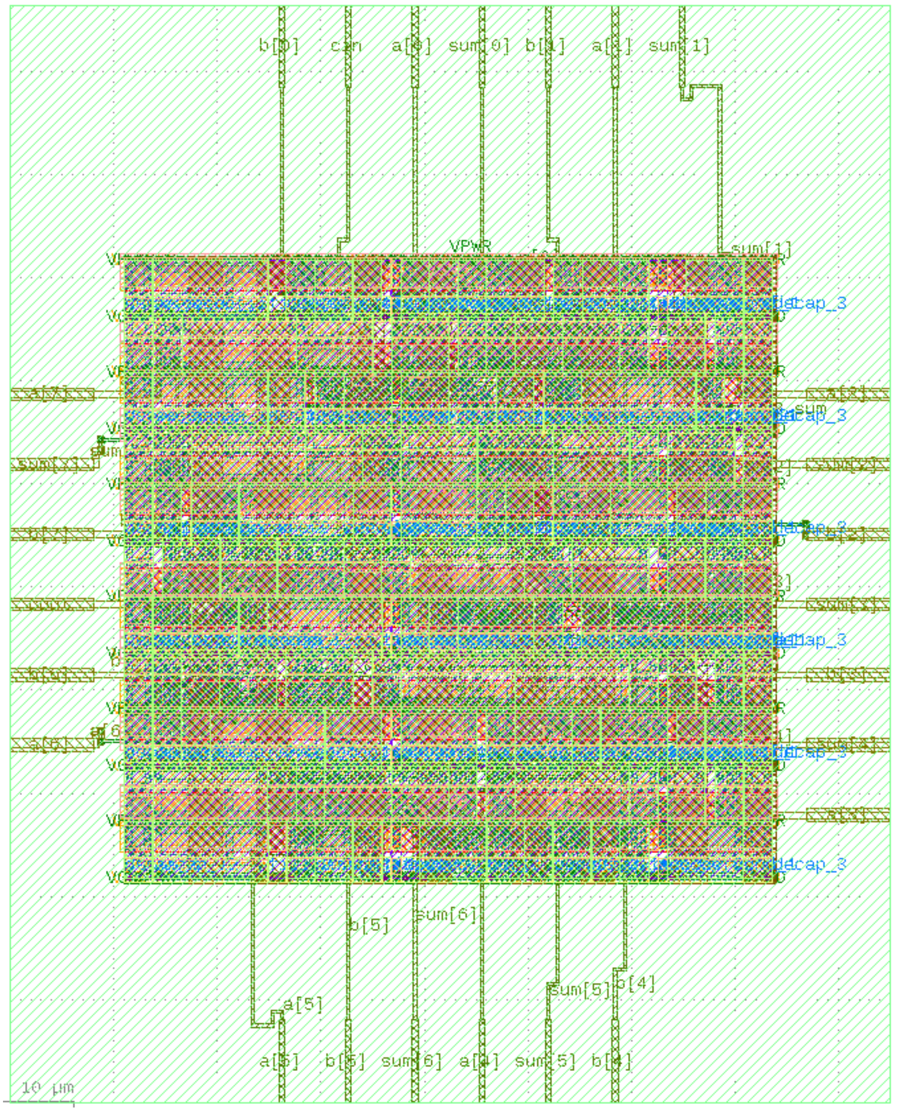
```
{
  "PDK": "sky130A",
  "PDKPATH": "/home/adi/Desktop/tools/open_pdks/sky130/sky130A/libs.ref/sky130_fd_sc_hd/lef",
  "STD_CELL_LIBRARY": "sky130_fd_sc_hd",
  "SCLPATH": "/home/adi/Desktop/tools/open_pdks/sky130/sky130A/libs.ref/sky130_fd_sc_hd/lef",
  "DESIGN_DIR": "/openlane/designs/carryskip/src",
  "DESIGN_NAME": "carryskip",
  "VERILOG_FILES": [
    "/openlane/designs/carryskip/src/carryskip.v",
    "/openlane/designs/carryskip/src/carryskip4.v",
    "/openlane/designs/carryskip/src/fulladder.v",
    "/openlane/designs/carryskip/src/mux.v"],
  "CLOCK_PORT": "",
  "CORE_AREA": 800,
  "DIE_AREA": 900,
  "CLOCK_PERIOD": 10.0,
  "FP_PDN_MULTILAYER": 1,

  "PL_TARGET_DENSITY": 0.65,
  "FP_CORE_UTIL": 42,
  "SAVE_FINAL_REPORTS": 1,
  "PL_ENABLE_CONGESTION_MAP": 1
}
```

Area = 0.0022608929000000003 mm^2

Conjustion = 8.39

Wire length = 758nm

Power = 2.35e-05

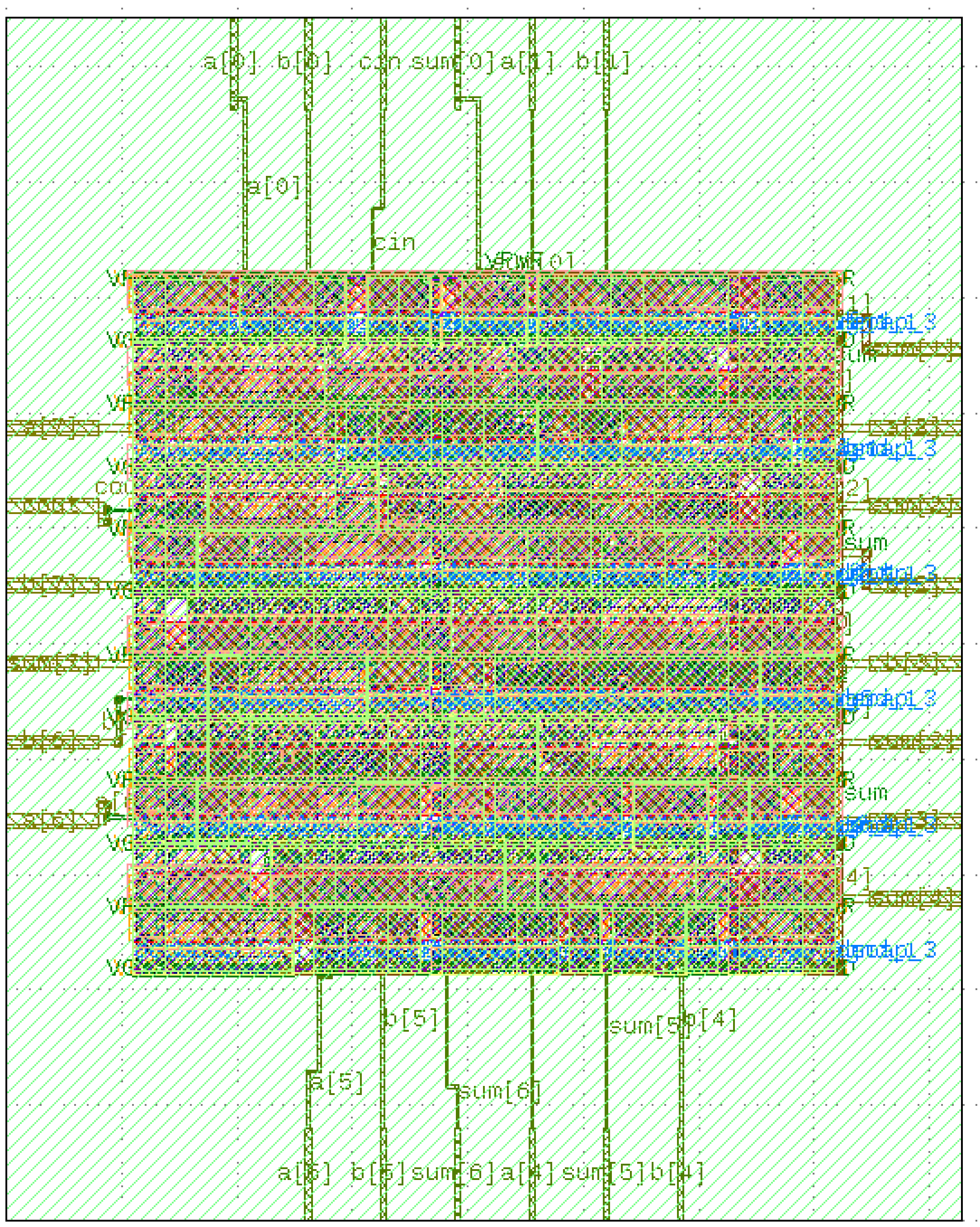


Now for the optimised we had to round off to the nearest whole number series


```json
{
  "PDK": "sky130A",
  "PDKPATH": "/home/adi/Desktop/tools/open_pdks/sky130/sky130A/libs.ref/sky130_fd_sc_hd/lef",
  "STD_CELL_LIBRARY": "sky130_fd_sc_hd",
  "SCLPATH": "/home/adi/Desktop/tools/open_pdks/sky130/sky130A/libs.ref/sky130_fd_sc_hd/lef",
  "DESIGN_DIR": "/openlane/designs/carryskip/src",
  "DESIGN_NAME": "carryskip",
  "VERILOG_FILES": [
    "/openlane/designs/carryskip/src/carryskip.v",
    "/openlane/designs/carryskip/src/carryskip4.v",
    "/openlane/designs/carryskip/src/fulladder.v",
    "/openlane/designs/carryskip/src/mux.v"],
  "CLOCK_PORT": "",
  "CORE_AREA": 600,
  "DIE_AREA": 650,
  "CLOCK_PERIOD": 10.0,
  "FP_PDN_MULTILAYER": 1,

  "PL_TARGET_DENSITY": 0.65,
  "FP_CORE_UTIL": 45,
  "SAVE_FINAL_REPORTS": 1,
  "PL_ENABLE_CONGESTION_MAP": 1
}
```


values so that we get optimised values

Area = 0.0021596388mm^2
Conjustion = 8.13
Wire length = 694
Power = 2.33e-05



Now to compare the percentage of Optimisation we can see that
Area - 4.48% dec
Power - 0.85% dec
Conjunction - 3.1% dec
Wire length = 8.4% dec