Phase 2

Objective:
Train an ML model to predict voltages at the lowest metal layer of the PDN. Train a machine learning model to predict the IR drop heat map.

Step 1: Data Collection
Use the SPICE netlists in the `training_data.zip` file to generate the dataset required for training your machine learning model. Run the solver developed in Phase 1 to collect this data. If your solver was inaccurate compared to SPICE, you can run SPICE simulations of all these netlists to collect the data and still be successful.

Step 2: Create Training Dataset (Features and Labels) :
The static IR drop distribution across the chip depends on the following three factors:

- The location/distributions of all voltage sources (power pads) in the design
- The topology of the PDN and resistance values of each resistor (via/metal layer) in the network
- The distribution of current sources (power) in the design

Prior ML techniques have modeled the above inputs as images (see example papers shared in the resources directory on Canvas) and used image-based ML models to predict IR drop.
Features: The example I presented in class represents the above three features as three distributions where the current source distribution is modeled as a current map (Fig. 1(a)); the PDN topology is modeled as a function of the density (spacing between power stripes per unit area) of the power grid (Fig. 1(b)) in different regions (there are only three types of densities), and the voltage source distribution is modeled as an effective distance map which is the distance from each PDN node to the PDN node with the voltage source (Fig. 1(c)). For example, one of the papers for Fig. 1(c), uses the idea of parallel paths and develops an effective distance to all N voltage sources as the reciprocal of the sum of distances to each voltage source.
Labels: The output is a distribution of IR drop at every node in the lowermost layer of the PDN, which can be represented as an IR drop map (Fig. 1(d)). Note that if you did not have good accuracy after Phase 1, you can use SPICE output directly as labels.

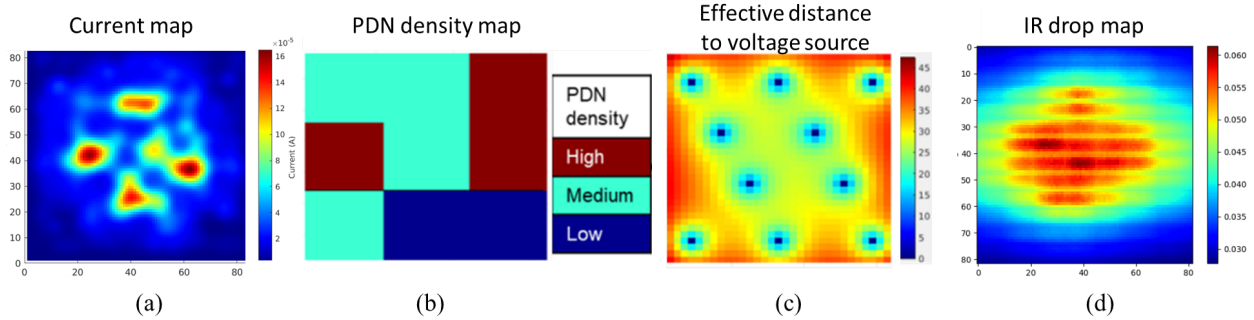


Fig. 1: Current map, PDN density, voltage source, and IR drop distributions across the chip.

In this step, your goal is to convert the SPICE netlist to the images shown in Fig. 1(a), (b), and (c). Then convert the output voltage file into an IR drop map based on the nodes located on the lowest layer of the PDN, as shown in Fig. 1(d). To convert your SPICE files and output voltage files from Phase 1 to images, you can use the x and y coordinates available in the node names.

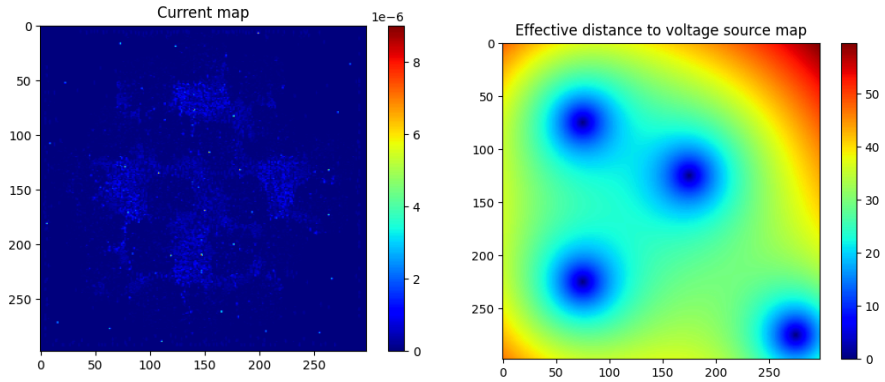
There are two ways in which you can do this:

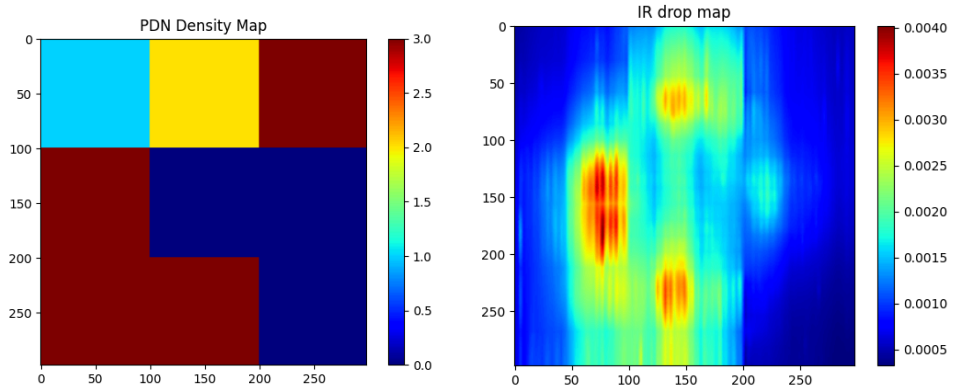
- Approach 1: Represent every 1μm² area of the location information encoded in the SPICE netlist into a single pixel. So if the area of the chip is 200um X 200um you will have images of sizes of 200x200. Note that if you choose this approach, you will have to ensure that your model can handle images of different sizes during both training and inference.
- Approach 2: Represent all testcases as images of the same size. This would lose information. Say a design of area 600um X 600um is the input test case, if they are represented as images of 256 x 256, then 1 pixel would represent 5.5μm² area, losing information.

Notes:

- In the SPICE netlists provided to you, a 2000 DBU has been used. In this case, you can divide all the values in the SPICE file by 2000 to get the exact location in micrometers.
- Note that these images are not RGB images. They are just 2D distributions of data. Do not map these into images like jpg, png, etc., with RGB. Work with these as 2D distributions of data or matrices.
- The runtime to create these 2D images is fast it takes me about few milliseconds to create them from the SPICE netlist.

Below is an example for testcase1.sp from your Phase 1 benchmarks.zip





Step 3:
Choose any ML model that can handle images, like CNN or U-Net. You can look at the literature and resource directory for inspiration on what ML models you want to work with.

Step 4:
Train the model using the dataset created in step 2 on the training_data.zip. There are ample resources available on the Internet for choosing and training a model using PyTorch or TensorFlow. I tested ChatGPT and it does a good job in creating a UNet in PyTorch and example training scripts. Try to understand the code and the architecture of the model so you can make modifications. Tune hyperparameters of the training, of the model, etc.

Step 5:
Use the 100 testcases in the training data.zip to train the model. Test the model by predicting the voltage IR drop map on the benchmarks.zip from Phase 1. Compare these with the golden voltage drop map (an image you create in Step 1). Use the following evaluation metrics to compare your accuracy:

However, the contestants' ML models will be evaluated on an unseen test dataset (real circuit data) and will be assessed for the following metrics:

(a) Mean absolute error (MAE): The average of the absolute difference between a prediction and the actual value, calculated for each benchmark. The goal is to have a low normalized MAE. This is the difference between the predicted image (2D distribution of IR drop) and the 2D distribution of the IR drop created from SPICE.

Example:
Predicted IR drop matrix is (hypothetical 2x2 image with 4 pixel values):

4.43E-03 5.83E-03

5.93E-03 1.04E-03

Actual IR drop matrix is (hypothetical 4x4 image with 4 pixel values):

4.63E-03 5.23E-03

5.93E-03 0.04E-03

MAE matrix is:

0.2E-03 0.6E-03

0 1E-03

MAE = 0.45mV

(b) F1 score: A binary classification metric that uses 10% of the maximum IR drop of each benchmark as the threshold to perform classification.

$$F1\ score = 2 * Precision * Recall / (Precision + Recall)$$
$$Precision = TP / (TP + FP)$$
$$Recall = TP / (TP + FN)$$

Where TP = True positive, FP = False positive, TN = True negative, and FN = False negative. The positive class is nodes with the top 10% of IR drop. The goal is to have a high F1 score. An element in the IR drop map is considered as an IR drop hotspot if its value is greater than 90% of the maximum IR drop of that benchmark. For the above example in (a), we define a hotspot matrix as an element that has an IR drop value larger than 90% of 5.93mV = 5.337mV. Please note that the maximum value will be different for each testcase

The actual hotspot matrix for the actual IR drop matrix above in (a):

0 1

1 0

The predicted hotspot matrix for the predicted IR drop matrix above in (a):

0 0

1 0

Therefore, TP = 1; TN = 2; FP = 0; FN = 1
$Precision = TP / (TP + FP)$ = 1/(1) = 1;
$Recall = TP / (TP + FN)$ = 1/(2) = 0. 5;
$F1\ score = 2 * Precision * Recall / (Precision + Recall)$ = 1/1.5 = 0.67

(c) Run time: Inference time of the ML model. The goal is to have low runtime and a fast inference.

Submission Process:

Your submission must include your Python scripts, which perform training and testing of the ML model. The top-level script must be named ir_predictor.py, and must have options to perform both training and inference as shown below:

```
python3 ir_predictor.py –training <path to folder containing spice
from training_data.zip and corresponding output files from phase 1>

python3 ir_predictor.py –inference  <path_to_test spice file>
```

Please create a requirements.txt similar to the one you created in Mini Project 1. You also need a README to explain how to run your code.

You also need to submit a report. The report should include the following details:

1) Features used
2) Choice of model details and reasons for it
3) A table that includes the above metrics for all the testcases in the benchmarks.zip folder