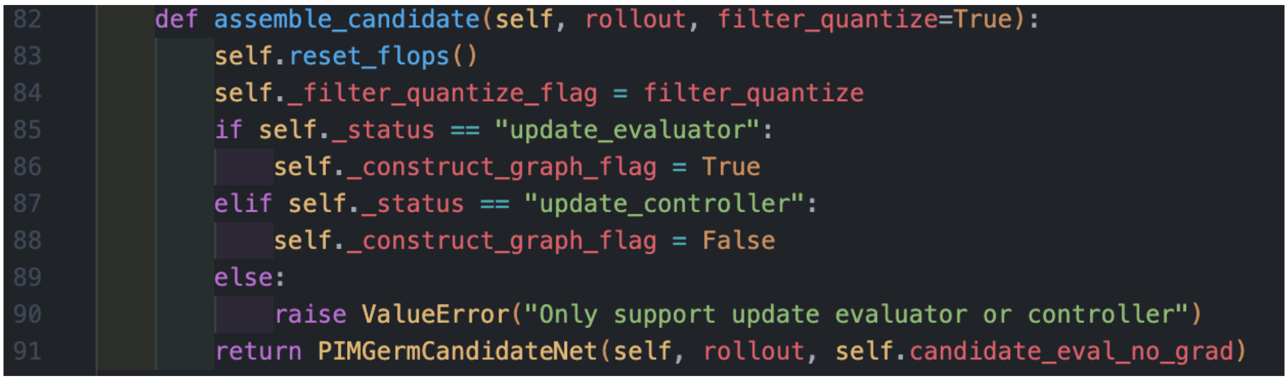
How to embed MNSIM in the NAS process

NAS framework

We recommend aw_nas as the Neural Architecture Search (NAS) framework. In aw_nas, you can utilize WeightsManager as the super net in the NAS process. A super net is a neural network that ensembles all neural network candidates and their shared weights. And different neural network candidates can be generated from the super net by different network genotypes (called rollout).

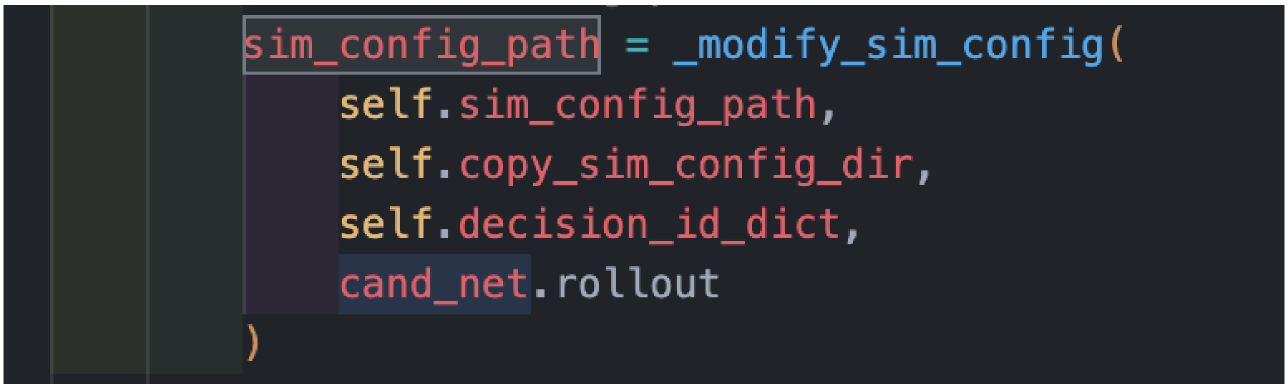
Embedding MNSIM in aw_nas

1. Sampling a candidate from the super net To test the hardware performance of a neural network candidate based on MNSIM, you need to sample a design candidate from the super net. Based on the sampled rollout, you can generate a candidate net (cand_net) as follows:



```python
    def assemble_candidate(self, rollout, filter_quantize=True):
        self.reset_flops()
        self._filter_quantize_flag = filter_quantize
        if self._status == "update_evaluator":
            self._construct_graph_flag = True
        elif self._status == "update_controller":
            self._construct_graph_flag = False
        else:
            raise ValueError("Only support update evaluator or controller")
        return PIMGermCandidateNet(self, rollout, self.candidate_eval_no_grad)
```

2. Generate new hardware config The hardware config is also can be searched in the NAS process. Therefore, you should modify the hardware config based on the genotypes of the neural network candidates. We recommend to modify the base hardware config file to get the new hardware config file as follows:

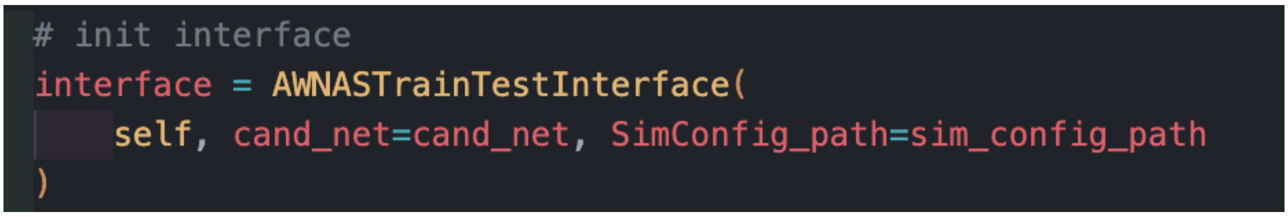


```python
sim_config_path = _modify_sim_config(
    self.sim_config_path,
    self.copy_sim_config_dir,
    self.decision_id_dict,
    cand_net.rollout
)
```

And the function goes as:

```python
def _modify_sim_config(simconfig_path, output_dir, decision_id_dict, rollout):
    # load origin sim config
    sim_config = cp.ConfigParser()
    sim_config.read(simconfig_path, encoding="utf-8")

    # modify crossbar size
    xbar_size = rollout[decision_id_dict["xbar_size"]]
    sim_config.set(
        "Crossbar level", "Xbar_Size",
        "{0},{0}".format(xbar_size)
    )
    # modify crossbar dac number and adc number
    DAC_GROUP = 4
    assert xbar_size % DAC_GROUP == 0
    sim_config.set(
        "Process element level", "DAC_Num",
        "{}".format(xbar_size // DAC_GROUP)
    )
    ADC_GROUP = 32
    assert xbar_size % ADC_GROUP == 0
    assert xbar_size % DAC_GROUP == 0
    sim_config.set(
        "Process element level", "ADC_Num",
        "{}".format(xbar_size // ADC_GROUP)
    )

    # modify input bit and quantize bit
    input_choice = rollout[decision_id_dict["dac_choice"]]
    sim_config.set("Interface level", "DAC_Choice", str(input_choice))
    output_choice = rollout[decision_id_dict["adc_choice"]]
    sim_config.set("Interface level", "ADC_Choice", str(output_choice))

    # modify weight bit
    device_level = 2**rollout[decision_id_dict["cell_bit"]]
    sim_config.set("Device level", "Device_Level", str(device_level))
    ## device resistance
    device_resistance = sim_config.get("Device level", "Device_Resistance").split(",")
    assert len(device_resistance) == device_level, \
        "Device resistance should be corresponding to device level"

    # write sim config
    output_path = os.path.join(output_dir, "sim_config.ini")
    with open(output_path, "w") as f:
        sim_config.write(f)
    return output_path
```

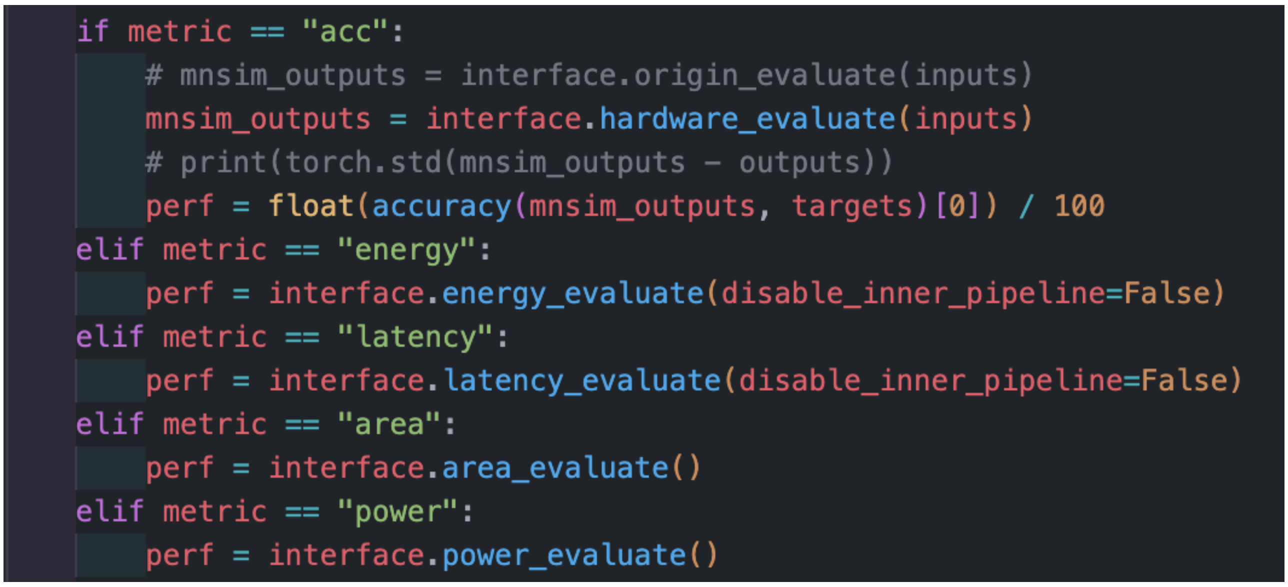
3. Initialize MNSIM Interface In this branch of the MNSIM, we add a new class called AWNASTrainTestInterface as the interface in the file ``MNSIM/Interface/awnas_interface.py'' between MNSIM and aw_nas. Based on the neural network candidate and the new hardware config, you can initialize the interface as follows:



```python
# init interface
interface = AWNASTrainTestInterface(
    self, cand_net=cand_net, SimConfig_path=sim_config_path
)
```

self is one type of the objective function in aw_nas

4. Get the hardware performance Now you can get the hardware performance based on MNSIM.



```python
    if metric == "acc":
        # mnsim_outputs = interface.origin_evaluate(inputs)
        mnsim_outputs = interface.hardware_evaluate(inputs)
        # print(torch.std(mnsim_outputs - outputs))
        perf = float(accuracy(mnsim_outputs, targets)[0]) / 100
    elif metric == "energy":
        perf = interface.energy_evaluate(disable_inner_pipeline=False)
    elif metric == "latency":
        perf = interface.latency_evaluate(disable_inner_pipeline=False)
    elif metric == "area":
        perf = interface.area_evaluate()
    elif metric == "power":
        perf = interface.power_evaluate()
```

- *hardware_evaluate* for the accuracy of the neural network candidate on the hardware
- *energy_evaluate* for the energy consumption of the neural network candidate on the hardware / mJ
- *latency_evaluate* for the latency of the neural network candidate on the hardware / ms
- *area_evaluate* for the area of the neural network candidate on the hardware / mm^2