RTSS 2025 CLARE Submission Artifact Evaluation Guide

Overview:

- A **docker image** is used for the artifacts, the URL to the docker image: https://hub.docker.com/r/jishixin/clare-sim/tags
- We uploaded all source codes of CLARE to the **GitHub repository**, URL: https://github.com/arc-research-lab/CLARE
    - The materials for artifact evaluation is in the branch `RTSS2025_AE`
- The manuscript is also uploaded to the GitHub repo, please refer to https://github.com/arc-research-lab/CLARE/blob/RTSS2025_AE/manuscript/rtss25-paper15.pdf
- Detailed explanation of source code can also be found in the Github repo
- We provide command line tools to reproduce figure 11(a), (b), (c), figure 13, and experiments we plan to add in the camera-ready manuscript.

Installation Guide

- The following sections are assuming the docker runs on Linux environments, the docker and the code is tested in Docker version 26.1.3, build 26.1.3-0ubuntu1~20.04.1
- If using windows setup, please refer to the Windows Setup subsection, still, a Linux environment is recommended
- `sudo` may be required for the docker image and container operations
- The docker image is originally build on Docker version 26.1.3, build 26.1.3-0ubuntu1~20.04.1

Install docker image

We provide 3 methods to install the docker image: pull from docker hub, download from Google Drive, and build from source

download docker image from docker hub(recommended)

```
docker pull jishixin/clare-sim:latest
```

download docker image from Google Drive

- As an alternative we also uploaded the docker image to Google Drive in `tar` format, the evaluator may download it and install the image
- The URL for docker image: https://drive.google.com/drive/folders/1Ksqt2EkWdDfyBeC58VYGKNtZvjASMVrL?usp=sharing
- Once the docker image is download, use the following command to install it:

```
docker load -i clare-sim.tar
```

- The compressed docker image is build from the following commands:

```
sudo docker save -o clare-sim.tar jishixin/clare-sim:latest
```

compile docker image from source code

- We also open source the docker file in the CLARE repo, users can reproduce the docker image based on the GitHub repo
- Clone the CLARE github repo

```
git clone https://github.com/arc-research-lab/CLARE.git
```

- Switch to the branch for RTSS 2025 AE

```
#within the CLARE repo
git checkout RTSS2025_AE
```

- Use docker build to build the image:

```
docker build -t jishixin/clare-sim:latest . #may require sudo
```

Verify docker image installation

- Check if the docker image is successfully installed:

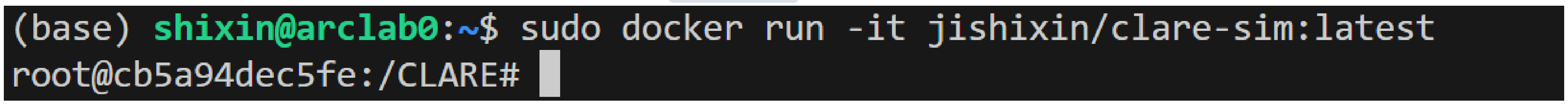
```
docker image ls |grep jishixin/clare-sim:latest
```

Launch the docker container

- After installing the docker, use the following command to launch a docker container for reproducing experiment
- No special requirements for the docker container

```
docker run -it jishixin/clare-sim:latest
```

- The shell terminal should then appear in `/CLARE` dir within the docker



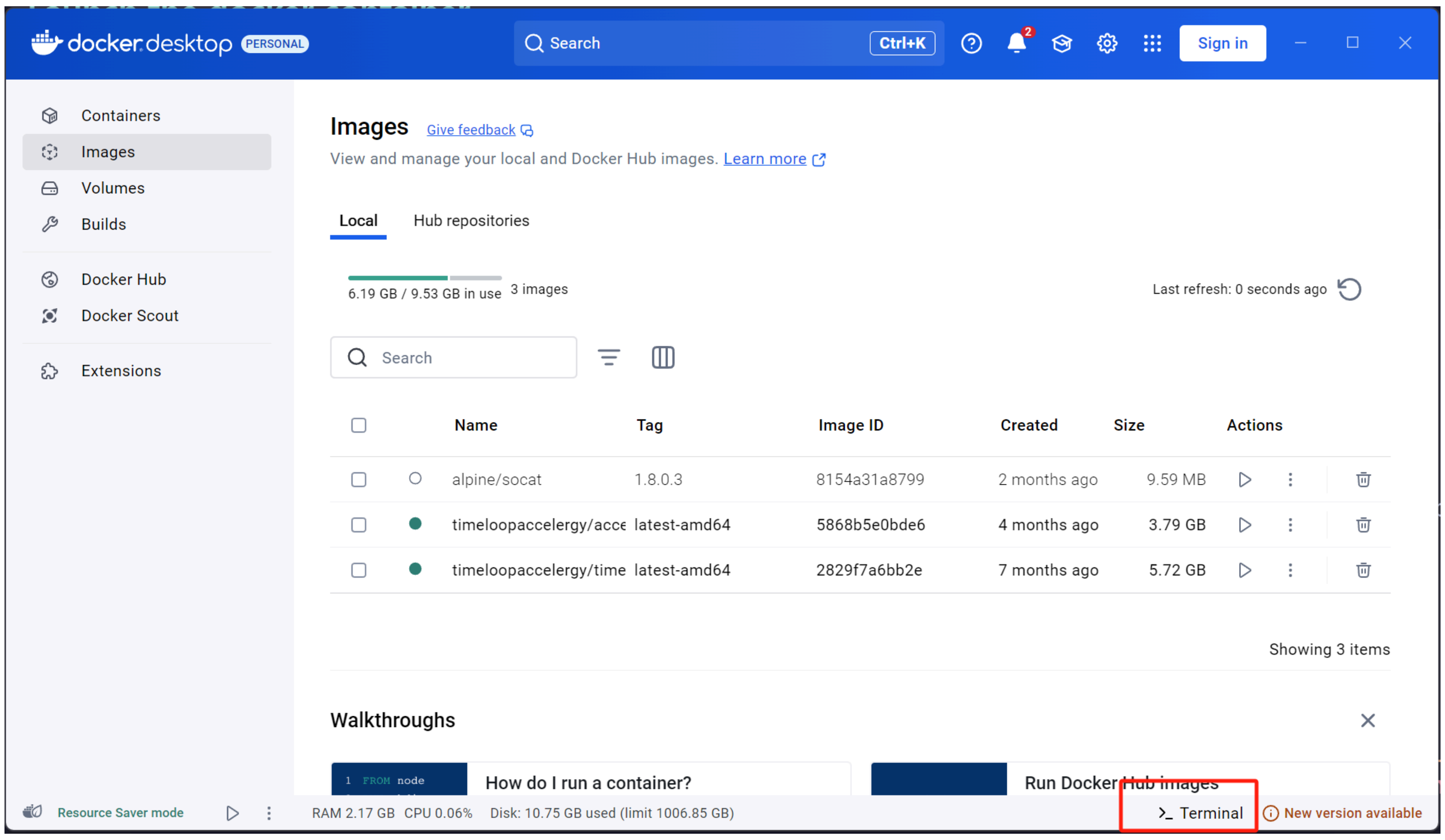
```
(base) shixin@arclab0:~$ sudo docker run -it jishixin/clare-sim:latest
root@cb5a94dec5fe:/CLARE#
```

Windows Setup using docker desktop

- The docker image and experiment passes the test on Windows 11 24H2 version 26100.4946 using docker desktop personal with Docker version 27.5.1, build 9f9e405

Install docker image

- open the docker desktop and the docker terminal



Images

View and manage your local and Docker Hub images. Learn more

6.19 GB / 9.53 GB in use 3 images

Last refresh: 0 seconds ago

Name	Tag	Image ID	Created	Size
alpine/socat	1.8.0.3	8154a31a8799	2 months ago	9.59 MB
timeloopaccelergy/acce	latest-amd64	5868b5e0bde6	4 months ago	3.79 GB
timeloopaccelergy/time	latest-amd64	2829f7a6bb2e	7 months ago	5.72 GB

Showing 3 items

RAM 2.17 GB CPU 0.06% Disk: 10.75 GB used (limit 1006.85 GB)

- Within the terminal, download docker image from docker hub

```
docker pull jishixin/clare-sim:latest
```

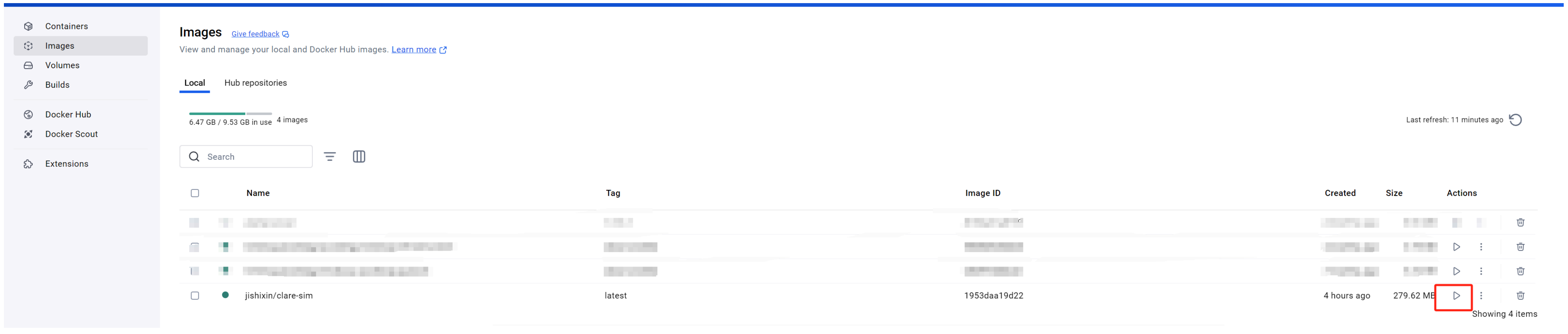
- launch a docker container

```
docker run -it jishixin/clare-sim:latest
```

- example terminal output:

```
PS C:\Users\jsx32> docker pull jishixin/clare-sim:latest
latest: Pulling from jishixin/clare-sim
b4d181a07f80: Pull complete
de8ecf497b75: Pull complete
baad0b8c8ed8: Pull complete
c4e37d93c85d: Pull complete
df93df0e898d: Pull complete
a4b5c2d8c17a: Pull complete
42ff5a723dd9: Pull complete
c4a7f8f897f5: Pull complete
8b03ab331280: Pull complete
Digest: sha256:42b14247869ad3a4e433dec03dc063c44f09d43cfd6d2bbbcd7723f4b49ae3a8
Status: Downloaded newer image for jishixin/clare-sim:latest
docker.io/jishixin/clare-sim:latest
PS C:\Users\jsx32> docker run -it jishixin/clare-sim:latest
root@9c1b7fa42350:/CLARE#
```

- alternatively, use the GUI to launch the container:

- 

Images

View and manage your local and Docker Hub images. Learn more

6.47 GB / 9.53 GB in use 4 images

Last refresh: 11 minutes ago

	Name	Tag	Image ID	Created	Size	Actions
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	
	jishixin/clare-sim	latest	1953daa19d22	4 hours ago	279.62 MB	

Showing 4 items

Step-by-step Guide on Reproducing CLARE Experiment Results

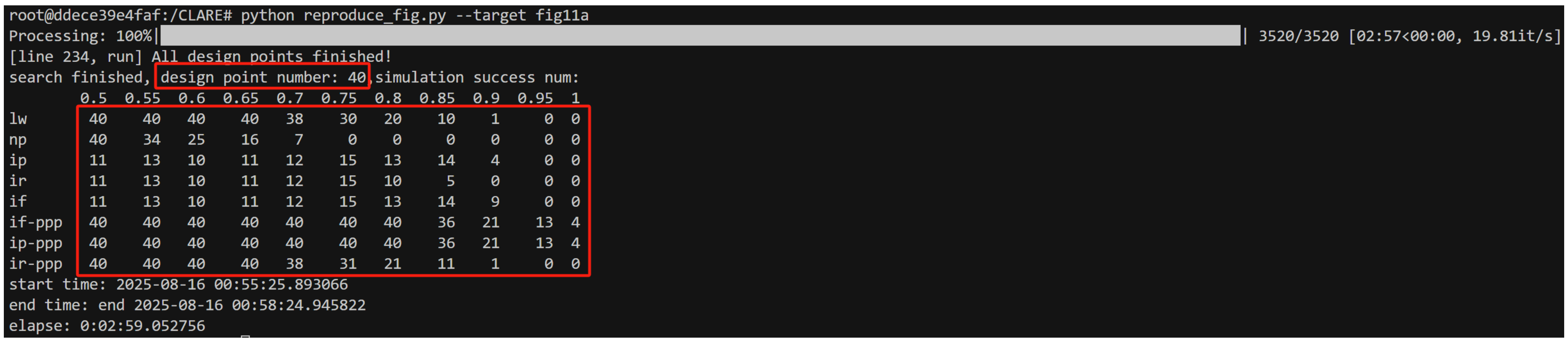
For more detailed explanations and instructions, please refer to the More Explanation on the Python Scripts section

Reproduce fig 11(a)

- Within the docker, use the following commands:

```
python reproduce_fig.py --target fig11a
```

- The simulation success number (i.e. the task meet the deadline) together with the total task number per (strategy,util) pair will be shown in the terminal, like:



```
root@ddece39e4faf:/CLARE# python reproduce_fig.py --target fig11a
Processing: 100%|████████████████████████████████████████████████████████████| 3520/3520 [02:57<00:00, 19.81it/s]
[line 234, run] All design points finished!
search finished, design point number: 40,simulation success num:
        0.5  0.55  0.6  0.65  0.7  0.75  0.8  0.85  0.9  0.95  1
lw       40    40   40    40   38    30   20    10    1     0  0
np       40    34   25    16    7     0    0     0    0     0  0
ip       11    13   10    11   12    15   13    14    4     0  0
ir       11    13   10    11   12    15   10     5    0     0  0
if       11    13   10    11   12    15   13    14    9     0  0
if-ppp   40    40   40    40   40    40   40    36   21    13  4
ip-ppp   40    40   40    40   40    40   40    36   21    13  4
ir-ppp   40    40   40    40   38    31   21    11    1     0  0
start time: 2025-08-16 00:55:25.893066
end time: end 2025-08-16 00:58:24.945822
elapse: 0:02:59.052756
```

Reproduce fig 11(b)

- within the docker, use the following commands:

```
#/CLARE#
python reproduce_fig.py --target fig11b
```

- results will be shown in terminal

Reproduce fig 11 (c)

- within the docker, use the following commands:

```
#/CLARE#
python reproduce_fig.py --target fig11c
```

- results will be shown in terminal

Reproduce fig 13

- within the docker, use the following commands:

```
#/CLARE#
python reproduce_fig.py --target fig13
```

- results will be shown in terminal

Reproduce additional experiment

- within the docker, use the following commands:

```
#/CLARE#
python reproduce_fig.py --target sche_vs_sim
```

More Explanation on the Python Scripts

CLARE Experiments overview

We conduct our experiments using simulation based on the fact that the latency of each operation on the FPGA accelerator is deterministic. We decouple the acceleration process of CLARE into different operations, profile each operation and conduct simulation based on these parameters.

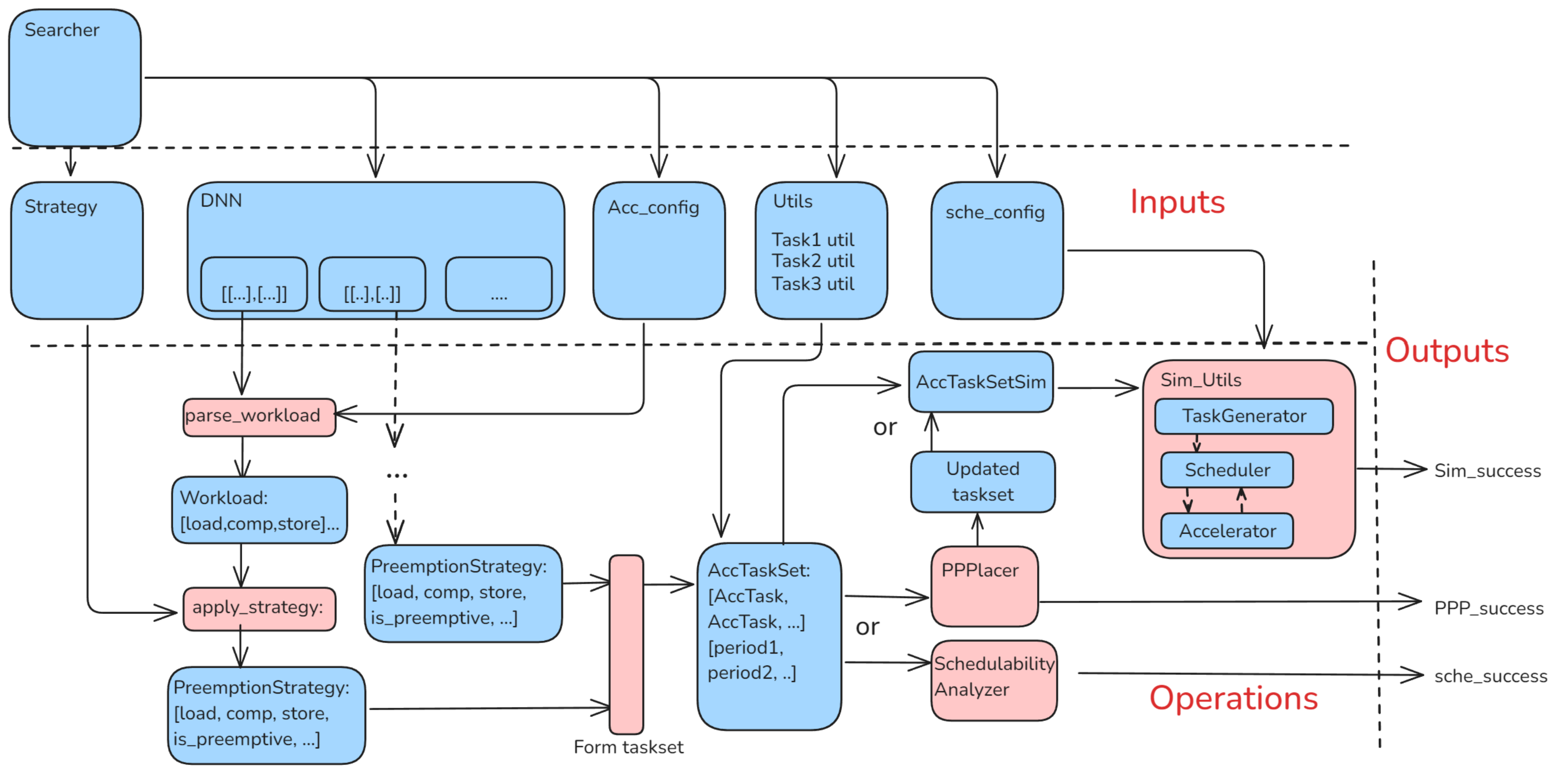
- The input parameters of accelerator are as shown in Table 1 & 2 in the manuscript, which is stored in the `CLARE/CLARE_SW/configs/acc_config.json`
- The input parameters of scheduler are as shown in Section 3C, which is stored in `CLARE/CLARE_SW/configs/sche_config.json`

CLARE focus on the matrix multiplication(MM) part of the accelerator, the input of DNN is a Python list of MM shapes. For example

```
DNN = [[128,256,128],[128,128,512]]
```

represents a DNN with two MM layers, where the shape is $A_{128,256} \times B_{256,128} = C_{128,128}$ and $C_{128,128} \times B_{128,512} = C_{128,512}$, respectively

A high-level description of CLARE experiments are as shown in the following diagram. For detailed documentation of the source code, please refer to the GitHub repo.



Input Strategy

The proposed CLARE intra-layer preemptive flexible dataflow enables more preemption points. We compare the proposed intra-layer preemptive schedule with the baseline schedule and conduct ablation studies, the available strategies include:

- non-preemptive (np)
- layerwise-preemptive (lw)
- intra-layer preemptive, recompute only, w/o preemption point placement(PPP) (ir)
- intra-layer preemptive, persist only, w/o PPP (ip)
- intra-layer preemptive, flexible, w/o PPP (if)
- intra-layer preemptive, recompute only, w/t PPP (ir-PPP)
- intra-layer preemptive, persist only, w/t PPP (ip-PPP)
- intra-layer preemptive, flexible, w/t PPP (if-PPP)

Input Utils

To ensure fair comparison, we set the total utilization of the taskset the same, then randomly generate utilization distributions for each task using the uunifast algorithm
One util distribution will be used for testing all strategies. In CLARE manuscript, we conduct more than 100 points for each strategy and total utilization.

Processing Flow for One configuration

- parse workload:
  - One input DNN model config will be parsed as an accelerator workload(`Workload()` `class`), containing iterations conducted by the accelerator given the fact that the tiling and dataflow of the accelerator is determined
  - each iteration has the information of (1)latency of load/compute/store (2)latency of swap-in/out operation when preemption happens (3) the location of this iteration, e.g. in the

    beginning or end of a layer

- apply strategy:
    - Based on the input strategies, the workload will be then converted to on child class of `PreemptionStrategy()`
    - This step determines the if a preemption can happen after each iteration (i.e., the preemption point)
    - This step also determines the strategy of implementing a preemption point(i.e., using recompute/persist, or this is the gap between layers thus no overhead is incurred)
- Form taskset:
    - First, each `PreemptionStrategy` object will be transferred as a `AccTask()` object:
        - Group the iterations in which preemption cannot happen as a `AccRegion(`), representing the non-preemptive region(NPR) in limited preemptive schedulability analysis
        - Compute the resume overhead($\xi_i, i > 1$) of each region using the swap-in overhead according to Equation 12 in the manuscript
        - Compute the execution length(exec time w/o preemption) and worst-case execution time(WCET)
    - All regions will form an `AccTaskSet()` with additional operations:
        - The period of each task is computed by the execution length and the utilization, then the taskset is sorted based on the utilization
        - The preemption overhead (ξ_1) is computed according to Equation 11 in the manuscript
        - The final taskset is compatible with the task setup as described in Section 4 A in the manuscript
- Schedulability analysis/ preemption point placement(PPP)
    - For the strategies does not require PPP, a `SchedulabilityAnalyzer()` is provided
        - The analyzer intakes the taskset, and determines if this taskset can pass schedulability analysis
    - For strategies requiring PPP, a `PP_placer` is provided
        - The place conducts PPP algorithm over the taskset by merging the NPRs
        - if the PP placement succeed, the taskset also passes the schedulability analysis, and a taskset after PPP will be returned and used
        - if not passing, the taskset also fails schedulability analysis, the original taskset will be used in future steps since no PPP solution is generated
- Simulation
    - The taskset will be first dumped to `AccTasksetSim`, which only contains the execution and preemption information of each NPR for performance
    - A `simpy` based simulator is provided, including:
        - `TaskGenerator`: release tasks periodically
        - `Scheduler`: performs limited preemptive EDF scheduling, also pays the scheduling operation latencies

    - `Accelerator`: simulates the accelerator behavior using the data within each NPR
  - The longest simulation time is set to the least common multiple(LCM) of all tasks in the taskset to ensure correctness

Searcher

- We provide a searcher class to conduct the experiment efficiently:
  - using the same DNN workload of one taskset, acc and sche configs, the searcher will automatically generate the configs of different total utilizations and strategies
  - multi-processing is used analyze different configs in parallel to enhance the performance.

Python command line tool

We provide python command line tool within the `reproduce_fig.py` file:

```
/CLARE# python reproduce_fig.py --help
usage: reproduce_fig.py [-h] --target {fig11a,fig11b,fig11c,fig13,sche_vs_sim} [--
size {small,large}]

cmd tool for reproduce CLARE RTSS2025 Submission figures

optional arguments:
  -h, --help            show this help message and exit
  --target {fig11a,fig11b,fig11c,fig13,sche_vs_sim}
                        The figure data to reproduce
  --size {small,large}  size of the experiment
```

- target: the corresponding data to reproduce.
- size: the scale of experiment, we provide small and large settings for each experiment. If the based computer is a PC or laptop, a small experiment is recommended.
- As a reference, the time of running a large experiment may take ~20 minutes in a 64-core CPU server.

Another small command line tool to print the excel files to terminal in a formatted manner. We use xlsx files to store the results for different applications

```
python print_xlsx.py <path_to_xlsx_file>
```

Scale of the experiment and performance issue

- Multi-processing is used to conduct analysis in parallel
- The schedulability analysis, PPP, and simulation are CPU-heavy tasks, please consider using a desktop PC or a CPU server, running experiment on laptop could be slow

experiment scales

- We provide small and large setting for each experiment
  - small settings uses reduced utilization(5% as a step) and number of design points(40) for each (utilization, strategy) setup, which is smaller than the setup in the manuscript but maintain the trend
  - large setting uses the same utilization(2.5% as a step) and number of design points(100) which is compatible with the setup in the manuscript. The random seed settings could be different, which may lead to slightly difference, whereas the key points are the same.
- By default, the command line tool use `small` setting. Users may add `--size large` to reproduce the full scale experiment

reference execution time

- Since random generated tasksets, even have the same total utilization
- 32-core, 64-thread CPU server with 2 Intel Xeon Gold 6346 CPUs
  - small setup: ~3 min per experiment
  - large setup: ~18 min per experiment
- 20-core, 20-thread Desktop PC with 1 Intel Ultra 7 265 CPU
  - small setup: ~5 min per experiment
- 8-core, 16-thread Desktop PC with 1 AMD Ryzen 7 7800X3D CPU
  - small setup: ~5 min per experiment
  - large setup: ~30min per experiment

temporary file structure

- The command line tool will create temporary files in the `/CLARE/temp` folder, keeping the generated input configs, the outputs and the results for each experiment
- The naming convention of the temp files:

```
/CLARE/temp/<target>_<size>/
```

- If a folder already exists in the path, the command line tool will try to use the kept inputs within this folder. The user may want to remove the temp files if they want to change to a new setup, or the temp file is corrupted

```
rm -r ./temp
```

Reproduce fig 11(a)

- Fig. 11(a) represents the success rate on synthetic workloads lean to the persist strategy

command line tool

- Use the command line tool to launch simulation:

```
python reproduce_fig.py --target fig11a
```

setup

- The default DNN setup are as follows:

```
DNN = [
        [[1024,8192,1024],[1024,8192,1024]],
        [[1024,8192,1024],[1024,8192,1024]]
    ]
```

Which means there are 2 same MLP models in the task set with the same shape, the first one has 2 MM layers, which each MM has the shape 1024x8192x1024 in the M,K,N dimension

- The default utilization and number of design points:

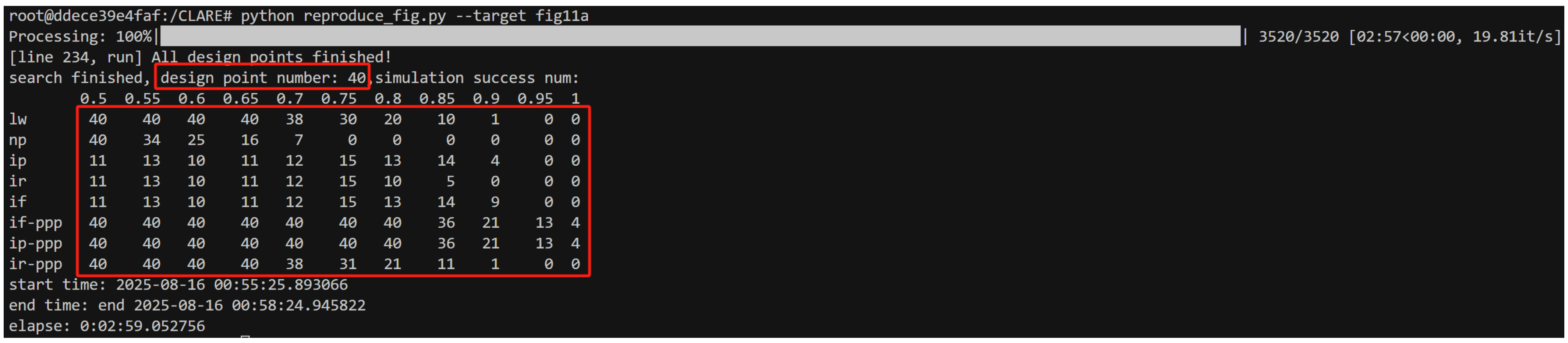
```
util_list_large = [0.5, 0.525, 0.55, 0.575, 0.6, 0.625, 0.65, 0.675, 0.7, 0.725, 0.75, 0.775, 0.8, 0.825, 0.85, 0.875, 0.9, 0.925, 0.95, 0.975,1] #for large exp
util_list_small = [0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8,0.85, 0.9, 0.95,1] #for small exp
num_util_large = 100 #for large exp
num_util_small = 40  #for small exp
```

temp files

- The results of schedulability analysis success rate, PPP success rate, and simulation success rate are saved as `.xlsx` file to the `/temp/fig11a_<size>` folder.

result format

The simulation success number (i.e. the task meet the deadline) together with the total task number per (strategy,util) pair will be shown in the terminal, like:



```
root@ddece39e4faf:/CLARE# python reproduce_fig.py --target fig11a
Processing: 100%|████████████████████████████████████████████████████| 3520/3520 [02:57<00:00, 19.81it/s]
[line 234, run] All design points finished!
search finished, design point number: 40,simulation success num:
        0.5  0.55  0.6  0.65  0.7  0.75  0.8  0.85  0.9  0.95  1
lw       40    40   40    40   38    30   20    10    1     0  0
np       40    34   25    16    7     0    0     0    0     0  0
ip       11    13   10    11   12    15   13    14    4     0  0
ir       11    13   10    11   12    15   10     5    0     0  0
if       11    13   10    11   12    15   13    14    9     0  0
if-ppp   40    40   40    40   40    40   40    36   21    13  4
ip-ppp   40    40   40    40   40    40   40    36   21    13  4
ir-ppp   40    40   40    40   38    31   21    11    1     0  0
start time: 2025-08-16 00:55:25.893066
end time: end 2025-08-16 00:58:24.945822
elapse: 0:02:59.052756
```

For example, the final success rate of non-preemptive strategy(np) at total util = 0.9 is 1/40 = 2.5%, whereas the intra-layer preemptive strategy reaches 21/40 = 52.5%

Reproduce fig 11(b)

- Fig. 11(b) represents the success rate on synthetic workloads lean to the recompute strategy

command line tool

- Use the command line tool to launch simulation:

```
python reproduce_fig.py --target fig11b
```

setup

- The default DNN setup are as follows:

```
DNN = [
        [[2048,128,2048],[2048,128,2048]],
        [[2048,128,2048],[2048,128,2048]]
    ]
```

- utilization and number of design points setup is same as fig 11(a)

temp files

- The results of schedulability analysis success rate, PPP success rate, and simulation success rate are saved as `.xlsx` file to the `/temp/fig11b_<size>` folder.

Reproduce fig 11(c)

- Fig. 11(b) represents the success rate on real workloads

command line tool

- Use the command line tool to launch simulation:

```
python reproduce_fig.py --target fig11b
```

setup

- We choose 5 realistic workloads which have more complicated shapes than synthetic workloads, the corresponding functions generating the shapes are in the `/CLARE/CLARE_SW/utils.py`
- We conduct the experiments of each workload separately, then sum the results up to get the final success rate
- For each workload, the strategy list is the same and follows the setup in fig11a
- For each workload, the `num_util` for each task is divided by 5
    - e.g. when using small setup, there will be 8 different utilization distribution generated for non-preemptive strategy at total util=0.5

temp files

- The results for each application and the accumulated results are saved as `.xlsx` to the `/temp/fig11c_<size>` folder.

- The accumulated results are just under the `/temp/fig11c_<size>` folder
- The results for each application is under `/temp/fig11c_<size>/<application>` folder

result format

- when running the experiment, the terminal will first output the success rate for each task, then the accumulated result, the fig 11c is based on the accumulated result

Reproduce fig 13

fig 13 compares the worst-case execution time under different utilizations
NOTE: when number of tasks within the task increases, the LCM of the periods will increase hugely. It is recommended to use the small setup even on a server, since the large setup can take hours to finish

command line tool

- Use the command line tool to launch simulation:

```
python reproduce_fig.py --target fig11b
```

setup

- We compare the WCET of different strategies, since the first task(task w/t smallest period) is always merged during PPP and will not be preempted by others, we use the task with the largest period
- for the strategies do not need PPP, the WCET is returned directly after form the taskset
- for the strategies with PPP, the WCET is returned after PPP
  - if PPP succeed, the WCET of the merged task set will be returned
  - if failed, the WCET of original task set will be returned
- for the taskset setup:
  - small setup: the taskset is made by 2 tasks of the same shape, num_util=20
  - large setup: the taskset is made by 3 tasks of the same shape, num_util=20

temp files

- The results for the average and normalized WCETs under different strategies and total utilizations are saved as `.xlsx` to the `/temp/fig13_<size>` folder.

result format

- The results of average and normalized WCETs are also output to the terminal
- Please note that in some cases, 'ir-ppp'(recompute) will have a smaller WCET than the 'if-ppp'(flexible) strategy, this is because the 'if-ppp' here only represents one of the proposed heuristics: the Persist-inclusive strategy. The Final proposed flexible strategy shown in the

manuscript will choose one with smaller WCET from the 'ir-ppp'(Recompute-only) and the 'if-ppp'(Persist-inclusive) strategy.
- For details of the proposed heuristics, please refer to Sec 4E in the manuscript.

Reproduce additional experiment

command line tool

- Use the command line tool to launch simulation:

```
#/CLARE#
python reproduce_fig.py --target sche_vs_sim
```

Explanation

- This experiment compares the success rate result difference between the schedulability analysis/PPP and the simulation
- The schedulability analysis serves as a sufficient condition of meeting the deadline:
  - if a task set passes the analysis, it is guaranteed to meet the deadline
  - however, if a task set do not pass, it may still have the chance of passing the deadline
- We reuse the data from fig11c to conduct this comparison, it is recommended to run the fig11c experiment first, then this additional experiment

Expected results

- The terminal will read and print the success rate of schedulability analysis and the simulation first, then show the difference
- It is expected the difference number is very small. It is OK that in small or even large setup, the difference is all zeros.
- This small gap between analysis and the simulation demonstrate the effectiveness of the schedulability analysis and our proposed heuristic, which represent a tight bound to the realistic scenarios

Documentation and Unit Tests

- A documentation for components of the CLARE software stack is in the README inside `CLARE_SW`
- We have also added detailed comments within each python script source code
- Some unit test of synthetic workload are provided in the bottom of each python script in `CLARE_SW` in the `if __name__ == '__main__'` branch, which is used to test the correctness in each step

Evaluation for CLARE hardware architecture

The CLARE hardware accelerator is also open-sourced and will be uploaded to the GitHub repo. Upon the evaluator's request, we can provide the HLS source code and generated bitstream of the CLARE accelerator. We can also provide the Vitis compilation environment for compiling the source codes and the AMD Versal VCK190 board as the testbed for on-board testing.