

# NeuStream Artifact Evaluation

The code repos are shared through google drive. The link is:

If the link is not valid, please check the <https://github.com/Fjallraven-hc/NeuStream-AE> to see valid link in README.md.

## 1. Diffusion

- Models:
  - Stable Diffusion v1.5
    - We test SD on both RTX4090 and H100, with 256x256 & 512x512 image generation.
  - Diffusion Transformer
    - We test origin DiT with its smallest (DiT\_S\_2) and biggest model (DiT\_XL\_2) size on 256x256 image generation task.
  - Palette
    - We test Palette on image restoration task on RTX 4090.
- Experiments
  - The dependency for Stable Diffusion and Diffusion Transformer is in **Diffusion/SD\_DiT.yaml**, the dependency for Palette is in **Diffusion/Palette.yaml**.
  - There are **run\_clockwork.sh** and **run\_neustream.sh** scripts in each subfolder.
  - The goodput data is extracted from the serving log, you can see **Diffusion/plot\_high\_goodput.py** and **Diffusion/plot\_low\_goodput.py** for the data in Figure 11 and Figure 12. The data for plot locates at the last line of the serving log in each subfolder, like below.

```
6211 key: UnetModule_send_time, value: 1714341548.993954
6212 key: VaModuleSafetyModule_receive_time, value: 1714341549.0606968
6213 key: VaModuleSafetyModule_send_time, value: 1714341549.1967993
6214 key: finish_time, value: 1714341549.2148443
6215 collector worker receive workload request count = 444
6217
6218 Server-side end2end latency: 2.757693298710449
6219 request step: 45key: guidance_scale, value: 7.5
6220 key: request_time, value: 1714341547.2325358
6221 key: sLO, value: 4.189420780915334
6222 key: id, value: 492
6223 key: ClipModule_receive_time, value: 1714341547.2338425
6224 key: ClipModule_send_time, value: 1714341547.2456498
6225 key: UnetModule_receive_time, value: 1714341547.2504973
6226 key: UnetModule_send_time, value: 1714341549.7838027
6227 key: VaModuleSafetyModule_receive_time, value: 1714341549.8218893
6228 key: VaModuleSafetyModule_send_time, value: 1714341549.9371996
6229 key: finish_time, value: 1714341549.9902291
6230 collector worker receive workload request count = 445
6231
```

```
742 request: 482, latency: 1.582458599809576
743 @Worker --- good : 359 --- finish: 381 --- abandon : 0 --- batch size : 1 --- ed : 1714
744 request: 485, latency: 1.21725252460812
745 request: 487, latency: 1.1128022164817
746 @Worker --- good : 381 --- finish: 383 --- abandon : 0 --- batch size : 2 --- ed : 1714
747 request: 489, latency: 1.452502917301684
748 @Worker --- good : 382 --- finish: 384 --- abandon : 0 --- batch size : 1 --- ed : 1714
749 request: 489, latency: 1.65407872288884
750 @Worker --- good : 383 --- finish: 385 --- abandon : 0 --- batch size : 1 --- ed : 1714
751 request: 490, latency: 1.197387101314807
752 @Worker --- good : 384 --- finish: 386 --- abandon : 0 --- batch size : 1 --- ed : 1714
753 request: 491, latency: 1.512289142088043
754 @Worker --- good : 385 --- finish: 387 --- abandon : 0 --- batch size : 1 --- ed : 1714
755 request: 492, latency: 1.395333013010748
756 @Worker --- good : 386 --- finish: 388 --- abandon : 0 --- batch size : 1 --- ed : 1714
757 request: 493, latency: 2.43544047738555
758 @Worker --- good : 387 --- finish: 389 --- abandon : 0 --- batch size : 1 --- ed : 1714
759 request: 494, latency: 4.622637050402466
760 @Worker --- good : 388 --- finish: 390 --- abandon : 0 --- batch size : 1 --- ed : 1714
761 request: 495, latency: 1.433002043032724
762 @Worker --- good : 389 --- finish: 391 --- abandon : 0 --- batch size : 1 --- ed : 1714
```

## 2. LLM

- We use the OPT model family to test the workload. For convenience, we didn't include origin model parameters in the Artifacts. Because under our evaluation in Figure 14 & 15, we choose the co-locate settings, so the prefill and decode instance are located in same process, and the implementation transforms to the execution order of prefilling and decoding, as shown in paper's Figure 10.

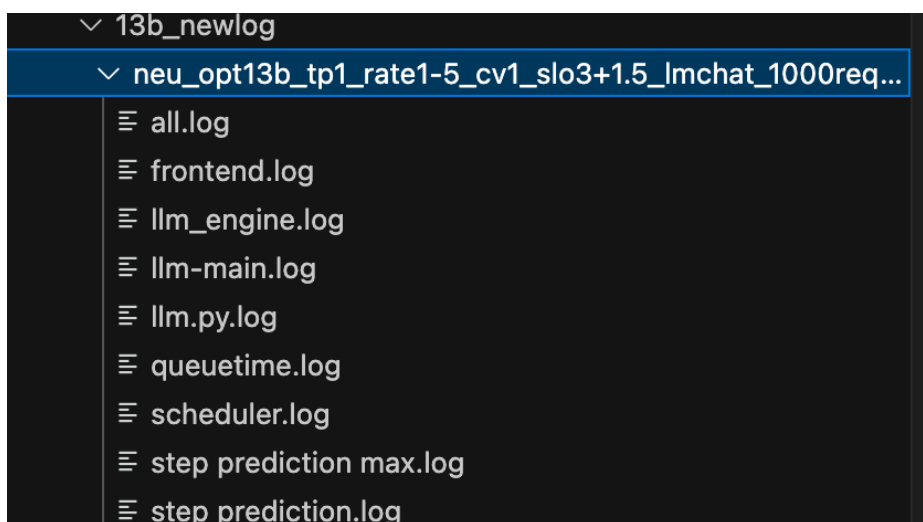
- To reproduce the data, it needs the NVIDIA A6000 and H100 accelerators.
- In the OPT-LLM folder, you can see three subfolders, **experiments** is for experiments, **neusim** is used for simulating the latency of prefill & decode execution when under different batch sizes or prefix lengths, and **vllm\_files** is the modified vLLM to support NeuStream in co-locating setting. To setup the environment, first create the conda env through the **LLM-OPT/experiments/conda.yaml**, and then install our modified vLLM backend in **LLM-OPT/vllm\_files/vllm-0.5.4**.
- **Experiments:**
  - To launch NeuStream, run **LLM-OPT/experiments/neurun.sh**
  - To launch origin vLLM, run **LLM-OPT/experiments/vllmrun.sh**
  - You can change the rate, cv, or slo through modifying the scripts, like below picture

```

all > exp > $ neurun.sh
1 # export $MODEL="facebook/opt-30b"
2 # export $TP_SIZE=1
3 # export $NEU_LOG_DIR="./logs/neu_rate"
4 # export CUDA_VISIBLE_DEVICES=0
5 # python poisson_experiment.py --gamma # --model $MODEL --tp_size $TP_SIZE --neu_dir $NEU_DIR
6 # DIR='./paper_results/13b/neu_opt13b_tp1_rate6.7_cv1-7_slo3+1.5_lmchat_1000req_seed0'
7 # DIR='./paper_results/66b/neu_opt66b_tp4_rate1.7_cv1_slo3+1.2-2.0_lmchat_1000req_seed0'
8 DIR='./paper_results/refine/neu_opt66b_tp4_rate2-5_cv1_slo1.5+1.5_lmchat_1000req_seedmap'
9 CUDA_VISIBLE_DEVICES=0,1,2,3 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma --rate 1,1.5 --cv 1 --pslo 1.5
10 # for rate in 5
11 # do
12 #   CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 0 -rws 2 --rate $rate
13 #   CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 1 -rws 2 --rate $rate
14 #   # CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 0 -rws 3 --rate $rate
15 #   # CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 1 -rws 3 --rate $rate
16 #   # CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 2 -rws 3 --rate $rate
17 # done
18

```

- The above command support multiple parameters in rate, cv or slo, but we recommend only passing one group multiple parameters.
- Model size and corresponding tensor parallelism need manually specification. Our search shows that the best tensor parallelism for OPT 13B, 30B, and 66B is 1, 2, and 4 respectively. NUM DEDUP is the number of repeated experiment groups in a setting, which can be set to 1 during the test. The final output log contains a step prediction max.log, which will output the highest goodput value in each of these experiments separately.



- Besides, when test different models, user has to manually specify the prediction/ simulation data for the certain model.

```

rate1-5_cv1_slo1.5-1.5_lmchat_100req_seedmap $ vllmrun.sh predictor.py X logger.py M sequence.py M llm.py M scheduler.py
vllm_files > vllm-0.5.4 > vllm > core > predictor.py > Predictor > __init__
58
59
60 class Predictor(PredictorBase):
61     def __init__(self, alpha):
62         super().__init__()
63         self.alpha = alpha
64         self.length_predict = (i:33-i if i < 33 else 1 for i in range(0, 10000))
65         # self.length_predict = (i:1 for i in range(0, 4096))
66         self.lenmeta = 32
67
68
69         # opt-13b tp=1 A6000
70         # self.prefill_time = np.array(
71         #     [0.04154373100027442, 0.04809862794354558, 0.04692623903974891, 0.048848932958208025, 0.06557490909472108, 0.06152643403038
72         # )
73         # self.timemodle_para_p = (0.013511968077884007, 0.0002467961652387274, -2.800508238190365e-09)
74         # self.timemodle_para_d = (0.04386336112795071, 1.0530766871910861e-06)
75
76         # opt-30b tp=2 A6000
77         # self.prefill_time = np.array(
78         #     [0.05193306994624436, 0.05726578098256141, 0.06202950899023563, 0.07150365295819938, 0.08614647993817925, 0.09257684892509
79         # )
80         # self.timemodle_para_p = (0.027307283178189653, 0.00032323153299157181, 8.706391148549e-09)
81         # self.timemodle_para_d = (0.054608089858371438, 9.136359951681938e-07)
82
83
84         # opt-66b tp=4 A6000
85         self.prefill_time = np.array(
86             [0.07411340903490782, 0.07838616904336959, 0.08576463698409498, 0.09083482890855521, 0.120361662004143, 0.12998087401501834,
87             )
88         self.timemodle_para_p = (0.03330899765431316, 0.0004740191671711913, 2.5876625483425424e-08)
89         self.timemodle_para_d = (0.07093051563015396, 7.632563111114172e-07)
90
91         self.pred_step_time = 0.01
92         self.pred_step = 100
93         self.pred_step_p = 100
94         # self.pred_gen_len = 80 #预测一个新的请求能生成多少token

```

### 3. Multi-Agent

- We use the MatPlotAgent(<https://github.com/thunlp/MatPlotAgent>) to test the workload. The environment need NVIDIA RTX4090 & H100 to reproduces the data in paper.
- Firstly, two models' parameter need to be downloaded.
  - <https://huggingface.co/llava-hf/llava-1.5-7b-hf>
  - <https://huggingface.co/meta-llama/CodeLlama-7b-Instruct-hf>

```

connect_final.py 3 M X
MatPlotAgent > connect_final.py > get_model
48 def frontend(input_queue: mp.Queue, output_queue: list[mp.Queue]):
49     output_queue[0].put(None)
50
51 def get_model(input_queue: mp.Queue, output_queue: mp.Queue, model_id, config):
52     model_class = config["model_config"][model_id]["class_name"]
53     other_params = {}
54     if "code" in model_class:
55         return CodeModel(input_queue, output_queue, name=config["model_config"][model_id]["name"], other_params
56     elif "llm" in model_class:
57         name = config["model_config"][model_id]["name"]
58         if "codellama" in name:
59             model_name = "/data/xuh/CodeLlama-7b-Instruct-hf"
60             llm = LLM(
61                 model=model_name,
62                 tensor_parallel_size=1,
63                 gpu_memory_utilization=0.90,
64                 disable_log_stats=True,
65                 max_model_len=4096,
66                 name=name,
67                 enable_chunked_prefill=args.chunk,
68             )
69             params = SamplingParams(temperature=0, top_p=1, max_tokens=1000,)
70             other_params["psio"] = 2
71         else:
72             model_name = "/data/xuh/llava-1.5-7b-hf"
73             llm = LLM(
74                 model=model_name,
75                 tensor_parallel_size=1,
76                 name=name,
77                 disable_log_stats=True,
78             )
79             params = SamplingParams(temperature=0, top_p=1, max_tokens=1000,)
80             other_params["psio"] = 2
81             llm.llm_engine.scheduler[0].do_predict = config["do_predict"]
82             other_params["sampling_params"] = params
83             other_params["dsio"] = 2
84             return llm, other_params
85         else:
86             raise NotImplementedError
87
88 def model_running(input_queue: mp.Queue, output_queue: mp.Queue, model_id: int, config, frontend_queue: mp.Queue=None):
89     testt = [t[0] for t in test]
90     time_re = [loop / t for t in testt]
91     assert len(testt) == test_num, "test re error!"

```

- Then, modify the model path in **MatPlotAgent/connect\_final.py**, as shown below.
- Secondly, see README.md in MatPlotAgent for experiments detail.