

# NeuStream-AE

Artifact Evaluation for NeuStream: Bridging Deep Learning Serving and Stream Processing, EuroSys25.

```
@inproceedings{
  title={NeuStream: Bridging Deep Learning Serving and Stream Processing},
  author={Yuan, Haochen and Wang, Yuanqing and Xie, Wenhao and Cheng, Yu and Miao, Ziming
and Ma, Lingxiao and Xue, Jilong and Yang, Zhi},
  booktitle={Proceedings of the Twentieth European Conference on Computer Systems},
  year={2025}
}
```

The experiments consist of three main parts: Diffusion, LLM, and Multi-Agent. For each part, we simulate a serving workload, send bulk requests to deployed modules through cross-process queues, and measure the system performance by goodput(the number of served requests that meet their SLO requirements per unit of time.).

## 1. Environment setup

We have set up the environments by conda on a RunPod cloud server with 4x RTX 4090 GPUs, we also downloaded the model parameters needed by the experiments.

```
conda activate Diffusion # Diffusion environment
conda activate LLM-OPT # LLM environment
conda activate MatPlotAgent # Multi-Agent environment
```

Below is how to set up the environments:

### 1. Diffusion environment setup:

1. `conda env create -f NeuStream-AE/Diffusion/Diffusion.yaml`
2. Comment out one line in installed diffusers library

```
# file_path: miniconda3/envs/Diffusion/lib/python3.12/site-
packages/diffusers/utils/dynamic_modules_utils.py

# from huggingface_hub import cached_download, hf_hub_download,
model_info
from huggingface_hub import hf_hub_download, model_info
```

### 2. LLM environment setup:

1. We modified vLLM to support NeuStream's temporal scheduling.

```
conda create -n LLM-OPT python=3.9 -y
cd /root/NeuStream-AE/LLM-OPT/vllm_files/vllm-0.5.4
pip install -e .
```

### 3. Multi-Agent environment setup:

1. See `NeuStream-AE/MatplotAgent/README.md` for details

## 2. Experiments

### 1. Diffusion experienments

1. The scripts to launch experiments are below

1. We have profiled data inside program to direct batching mechanism, so the results can be reproduced when launched on corresponding GPUs.

```
# DiT_S_2 256x256 image generation on RTX 4090
NeuStream-AE/Diffusion/DiT/RTX4090_DiT_S_2_img256/run_clockwork.sh
NeuStream-AE/Diffusion/DiT/RTX4090_DiT_S_2_img256/run_neustream.sh
# DiT_XL_2 256x256 image generation on RTX 4090
NeuStream-AE/Diffusion/DiT/RTX4090_DiT_XL_2_img256/run_clockwork.sh
NeuStream-AE/Diffusion/DiT/RTX4090_DiT_XL_2_img256/run_neustream.sh
# Palette 256x256 image restoration on RTX 4090
NeuStream-AE/Diffusion/Palette/run_clockwork.sh
NeuStream-AE/Diffusion/Palette/run_neustream.sh
# SD v1.5 256x256 image generation on RTX 4090
NeuStream-
AE/Diffusion/StableDiffusion/RTX4090_SD_FP16_img256/run_clockwork.sh
NeuStream-
AE/Diffusion/StableDiffusion/RTX4090_SD_FP16_img256/run_neustream.sh
# SD v1.5 512x512 image generation on RTX 4090
NeuStream-
AE/Diffusion/StableDiffusion/RTX4090_SD_FP16_img512/run_clockwork.sh
NeuStream-
AE/Diffusion/StableDiffusion/RTX4090_SD_FP16_img512/run_neustream.sh
# SD v1.5 512x512 image generation on H100
NeuStream-
AE/Diffusion/StableDiffusion/H100_SD_FP16_img512/run_clockwork.sh
NeuStream-
AE/Diffusion/StableDiffusion/H100_SD_FP16_img512/run_neustream.sh
```

1. Each experiment script will log the serving details in corresponding log file. The goodput is collected from serving log of different rate,cv, and slo. For example, compare logs

```
NeuStream-
```

```
AE/Diffusion/DiT/RTX4090_DiT_S_2_img256/neustream_rate_log_request500/2024-
04-28
```

```
17:41:55_Gamma_rate=4_cv=2_slo_factor=3.0_request=500_step_delta=0.95_device
=RTX4090_image_size=256_2024-04-28 17:41:55
```

and

```
NeuStream-
```

```
AE/Diffusion/DiT/RTX4090_DiT_S_2_img256/clockwork_rate_log_request500/2024-
04-28
```

```
17:47:55_Gamma_rate=4_cv=2_slo_factor=3.0_request=500_device=RTX4090_image_s
```

```
ize=256.log,
```

we will find that when serving a workload with rate=4qps, cv=2, and slo=3, NeuStream achieves 463 goodput while clockwork's algorithm achieves 86 goodput. These data points from logs make up the Figure 11 and Figure 12 in the paper.

2. The Figure 13 shows the batch size of DiT module when serving workload by NeuStream and Clockwork, it can be plotted by extract th DiT module size in log files like

```
NeuStream-
```

```
AE/Diffusion/DiT/RTX4090_DiT_S_2_img256/neustream_rate_log_request500/2024-04-28
```

```
17:41:55_Gamma_rate=4_cv=2_slo_factor=3.0_request=500_step_delta=0.95_device=RTX4090_image_size=256_DiTModule.log and NeuStream-
```

```
AE/Diffusion/DiT/RTX4090_DiT_S_2_img256/clockwork_rate_log_request500/2024-04-28
```

```
17:47:55_Gamma_rate=4_cv=2_slo_factor=3.0_request=500_device=RTX4090_image_size=256.log .
```

```
# For example
```

```
# Below log that shows Clockwork batch 3 requests at this round.
```

```
0号worker ---- good : 53 --- finish: 55 --- id: 290 309 310
```

```
# Below log shows NeuStream batch 10 requests at this round.
```

```
{"batch_size": 10, "time": 1714297483.5302887, "high_priority_count": 4, "low_priority_count": 6, "queue_size_before_schedule": 10, "batch_size_after_schedule": 0, "running_requests_id_list": [490, 491, 492, 498, 499, 497, 496, 495, 493, 494], "rest_time": [3.1086456775665283, 2.217283248901367, 3.7875068187713623, 2.854764223098755, 2.882814884185791, 3.5825355052948, 3.6029703617095947, 3.283489465713501, 2.5172903537750244, 3.5290353298187256]}
```

## 2. LLM-OPT experiments

1. We use the OPT model family to test the workload. For convenience, we didn't include origin model parameters in the Artifacts. 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, **exp** 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.
- **Experiments:**
  - To launch NeuStream, run **NeuStream-AE/LLM-OPT/exp/neurun.sh**

- To launch origin vLLM, run **NeuStream-AE/LLM-OPT/exp/vllmrun.sh**
- You can change the rate, cv, or slo through modifying the scripts, like below picture
- 

```

all > exp > $ neu-run.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(On NVIDIA A6000 and H100). NUM DEDUP is the number of repeated experiment groups in a setting, which can be set to 1 during the test.

```

248 # python poisson_experiment.py --gamma # --model $MODEL --tp_size $TP_SIZE --neu_dir $NEU_DIR
249 # DIR='./paper_results/13b/neu_opt13b_tp1_rate6.7_cv1-7_slo3+1.5_lmchat_1000req_seed0'
250 # DIR='./paper_results/66b/neu_opt66b_tp4_rate1.7_cv1_slo3+1.2-2.0_lmchat_1000req_seed0'
251 # DIR='./paper_results/refine/neu_opt66b_tp4_rate2-5_cv1_slo1.5+1.5_lmchat_1000req_seedmap'
252 # for rate in 5
253 # do
254 #     CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 0 -rws 2 --rate $rate
255 #     CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 1 -rws 2 --rate $rate
256 #     CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 0 -rws 3 --rate $rate
257 #     CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 1 -rws 3 --rate $rate
258 #     CUDA_VISIBLE_DEVICES=4,5 NEU_LOG_DIR=$DIR python poisson_experiments.py --gamma -rid 2 -rws 3 --rate $rate
259 # done
260
261 sampling_params = SamplingParams(temperature=0,
262                                 top_p=1,
263                                 max_tokens=args.max_tokens)
264 # model_name = "facebook/opt-30b"
265 # model_name = "facebook/opt-66b"
266 model_name = "facebook/opt-13b"
267 llm = LLM(model=model_name, tensor_parallel_size=1, gpu_memory_utilization=0.9, num_max_batched_tokens=8192, disable_log_stats=False)
268 llm.llm_engine.scheduler[0].do_predict = not vllm
269 predictor = llm.llm_engine.scheduler[0].predictor
270
271 with open("./inputs/longprompt2.txt") as f:
272     longpmp = f.readline()
273 llm.generate([longpmp, longpmp], sampling_params=SamplingParams(temperature=0, top_p=1, max_tokens=40))
274
275 # rates = [1, 1.5, 1.6, 1.9]
276 rates = [float(rt) for rt in args.rate.split(',') if args.rate else [4]]
277 rates.reverse()
278 dslos = [float(dslo) for dslo in args.dslo.split(',') if args.dslo else [1.5]]
279 pslos = [float(pslo) for pslo in args.pslo.split(',') if args.pslo else [3]]
280 pred_steps = [30]
281 pred_genlens = [None]
282 pred_step_times = [0.123]
283
284 cvs = [float(cv) for cv in args.cv.split(',') if args.cv else [1]]
285 if not args.gamma:
286     cvs = [1]
287 NUM_DUP = 5
288 # 放入循环的次数
289 queue_llm.put((len(rates)*len(pslos)*len(dslos)*len(pred_step_times)*len(pred_steps)*len(pred_genlens)*len(cvs)*NUM_DUP))
290
291 for pred_genlen, rate, pred_step_time, pslo, dslo, pred_step, cv in itertools.product(pred_genlens, rates, pred_step_times, pslos, dslos, pred_steps, cvs):
292     info = f"model: {model_name}, Schedule_enabled: {not args.vllm}, pslo: {pslo}, dslo: {dslo}, pred_step: {pred_step}, pred_step_time: {pred_step_time}, cv: {cv}"
293     llm.llm_engine.log(info)
294     logger2.debug(info)

```

- The final output log contains a step prediction max.log, which will output the highest goodput value in each of these experiments separately.
  - the file path is
- Besides, when test different models, user has to manually specify the prediction/simulation data for the certain model.

```
NeuStream-AE > LLM-OPT > vllm_files > vllm-0.5.4 > vllm > core > predictor.py
46 class PredictorOutput:
47     def __init__(self, output):
48         self.output = output
49
50 class Predictor(PredictorBase):
51     def __init__(self, alpha) -> None:
52         super().__init__()
53         self.alpha = alpha
54         self.length_predict = {i:33-1 if i < 33 else 1 for i in range(0, 10000)}
55         # self.length_predict = {i:1 for i in range(0, 4096)}
56         self.lenmeta = 32
57
58     # opt-13b tp=1 A6000
59     # self.prefill_time = np.array(
60     #     [0.04154373100027442, 0.04809862794354558, 0.04692623903974091, 0.048048932958208025, 0.05557490909472108, 0.061526434038
61     # ]
62     # self.timemodle_para_p = (0.813511068077884007, 0.0002467961652307274, -2.800509238190365e-09)
63     # self.timemodle_para_d = (0.04306336112795071, 1.0530766871910861e-06)
64
65     # opt-30b tp=2 A6000
66     # self.prefill_time = np.array(
67     #     [0.055193306994624436, 0.05726578098256141, 0.06202950899023563, 0.07150365295819938, 0.08614647993817925, 0.092576848925
68     # ]
69     # self.timemodle_para_p = (0.027307283178189653, 0.0003323153299157181, 8.706391148549e-09)
70     # self.timemodle_para_d = (0.05460008958371438, 9.136359951081938e-07)
71
72     # opt-66b tp=4 A6000
73     self.prefill_time = np.array(
74     [0.07411340903490782, 0.07838616904336959, 0.08576463698049498, 0.09083482890855521, 0.120361662004143, 0.12990807401834
75     ]
76     self.timemodle_para_p = (0.03330899765431316, 0.0004748191671711913, 2.5876625483425424e-08)
77     self.timemodle_para_d = (0.07093051563015396, 7.6325631111114172e-07)
78
79     self.pred_step_time = 0.01
80     self.pred_step = 100
81     self.pred_step_p = 100
82     # self.pred_gen_len = 80 #预测一个新的请求能生成多少token
83
84     self.decision_histoty: List[int] = []
85     self.step_time_history: List[float] = []
```

Because of the GitHub single file size limitation, user has to manually run the script

`merge_large_file_fragment.sh` to get the origin model parameters.

For convenience, we didn't upload unnecessary model parameters, so part of the models are serving with randomly initialized parameters, but this has no effect on the serving performance.

The artifacts will be also provided through Baidu Netdisk, where some model parameters are also stored.

The uncompressed file size is about 20GB.

For Diffusion and LLM part, please access the artifacts through <https://pan.baidu.com/s/19pzsF1lBwu7Qkj42j2R4ng?pwd=qq45> code:qq45

For MultiAgent part, please access the artifacts through <https://pan.baidu.com/s/1PyaLMXABYd0jcFV8raaYaA?pwd=mmmpc> code:mmmpc

Alternatively, artifacts can be accessed through OneDrive link: <https://1drv.ms/u/c/e3200eaf81ba2fa8/EfDf2yp2UEZMi9gdaVQ-1TMBn37em9XQRs9HwmL524s-JA?e=bxCCnC>