CSC 591 - GEN AI IN SYSTEMS UNITY ID- sgundew

Homework 1

Github link for the homework code

The GH Repo also contains python code for various plots created below .
Section 1:

Q1. The authors mention two approaches to improve cache performance (hit rate). Which approach does the paper focus on?

Ans: The author mentions below two approaches:

1. Prefetching – Guessing which data will be needed in future and loading it in the cache.
2. Cache Replacement – Selecting which data to evict from the cache to make space for the current data.

The paper focuses on the 2nd approach i.e. **Cache Replacement**.

Q2. What are the previous state-of-the-art approaches mentioned in the paper?

Ans:

1. Hawkeye
2. Glider
3. Heuristic based : LRU and MRU

Q3. Explain the cache hierarchy used for experimentation by the authors. This means you have to give the size of L1/L2/L3 caches including the number of sets and ways.

Ans: The cache hierarchy used in the experiments consists of three levels:

$$Number\ of\ sets = Cache\ Size/(associativity * Block\ Size)$$

1. L1 Cache: 32 KB, 4-way set-associative
   32×1024/(4×64) = 128 sets
2. L2 Cache: 256 KB, 8-way set-associative
   256×1024/(8×64)=512 sets
3. L3 Cache (Last-Level Cache): 2 MB, 16-way set-associative
   2×1024×1024/(16×64)=2048 sets

   The experiments focus on optimizing L3 cache replacement policies, while L1 and L2 caches use the LRU replacement policy.

Q4. What is the normalized cache hit rate? Why have the authors used it as a metric?

Ans: The normalized cache hit rate is defined as a metric that measures the effectiveness of a cache replacement policy relative to two baselines: the Least Recently Used (LRU) policy and Belady's optimal policy.

$$\text{Normalized Cache Hit Rate} = (r - rLRU)/ (ropt - rLRU) ,$$

Where,
r = raw cache hit rate current replacement policy
*r*opt = raw cache hit rate of Belady's optimal policy.
*r*LRU = raw cache hit rate of LRU policy.

The authors have used it as a metric because,

- Relative Performance: It quantifies how close a policy's performance is to the optimal (Belady's) and how much it improves over a baseline (LRU), providing a clear comparison.
- Fair Comparison Across Workloads: accounts for differences in cacheability across workloads, enabling consistent evaluation.
- Standardization: simplifies interpreting results by normalizing performance between two well-understood baselines (LRU and Belady's).

Q5. Interpret Figure 2 from the paper. Explain what you understand by looking at the figure and the caption.

Ans: Figure 2 in the paper shows the normalized cache hit rates of Belady's algorithm as a function of the size of the future window it uses to find the reuse distances.

- The x-axis shows the number of future accesses (window size) that Belady's algorithm considers when making a choice for eviction.
- The y-axis shows the normalized cache hit rate, where 0 is the performance of LRU (Least Recently Used), and 1 is the performance of Belady's with an infinite future window (optimal).
- As the future window size increases, Belady's algorithm achieves higher normalized cache hit rates.
- To reach 80% of the performance of Belady's optimal policy, it requires accurately computing reuse distances for cache lines up to 2600 future accesses.

It basically shows that we will need huge amounts of data to reach as close as possible to Belady's optimal solutions which the authors are trying to solve with their proposed PARROT solution.

Q6

Question VI) The following table shows the number of misses encountered while evaluating/testing the Parrot model in the second column. The last column estimates the total number of cache misses, i.e., the speculated number of cache misses for the complete trace and all cache sets. What formula might be used to calculate the values in the last column? Explain your answer.

A	B	C
Trace	# of misses (test)	Total # of misses
lbm_564B	95424	30535680
astar_313B	94178	30136960
milc_409B	91172	29175040

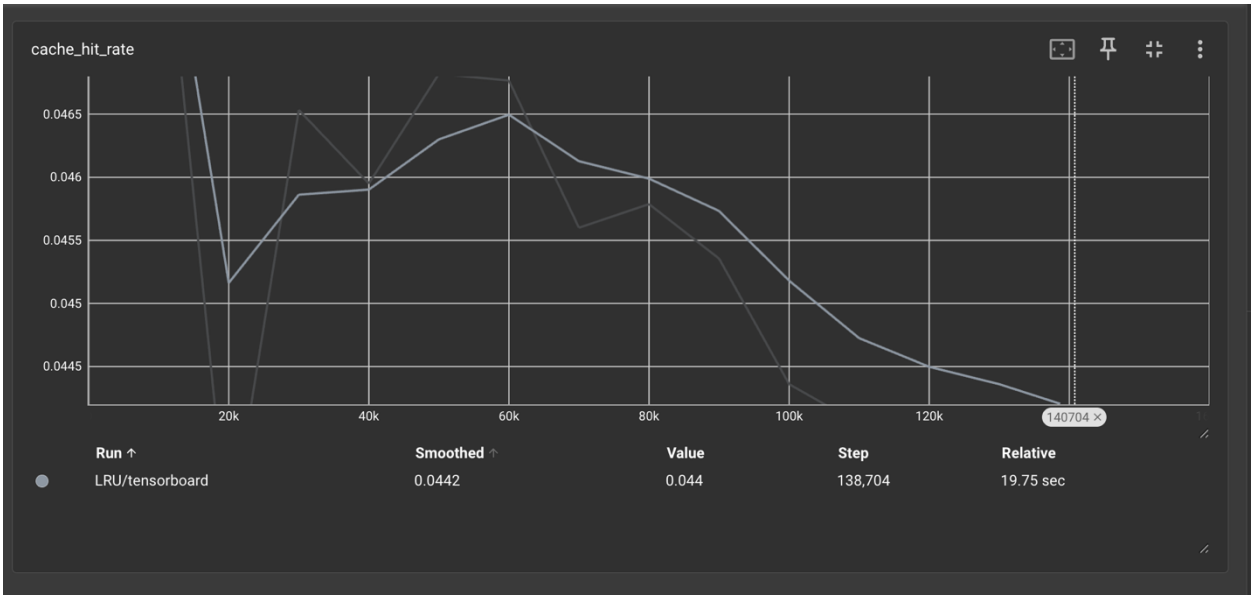
Ans:
If we Divide total # of misses by # of misses(test), in all 3 traces, the output comes to some sort of a scaling factor of 320.
Therefore we can estimate that the # total # of misses can be # test misses multiplied by the train-test split factor which is in our case is 320.

Section 2:

Task 1: LRU Policy

Bash script:

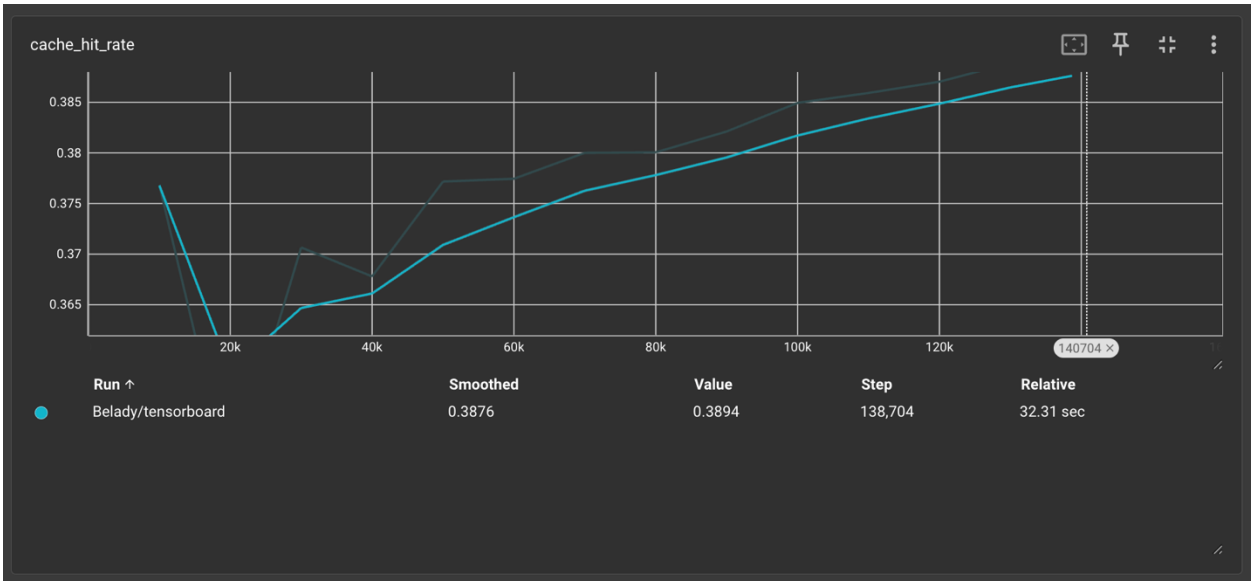
```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -q gpu
#BSUB -gpu "num=1"
#BSUB -J training_phase_lru
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/err.%J

source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP

python3 -m cache_replacement.policy_learning.cache.main \
--experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments \
--experiment_name=astar_lru \
--cache_configs=cache_replacement/policy_learning/cache/configs/default.json \
--cache_configs=cache_replacement/policy_learning/cache/configs/eviction_policy/lru.json \
--memtrace_file=/share/$GROUP/traces/astar_313B_test.csv
```



Task 2: Belady

Bash Script: Belady Policy

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -q gpu
#BSUB -gpu "num=1"
#BSUB -J training_phase
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/LSTM/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/LSTM/logs/err.%J



source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/LSTM



python3 -m cache_replacement.policy_learning.cache.main \
--experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments \
--experiment_name=astar_belady \
--cache_configs=cache_replacement/policy_learning/cache/configs/default.json \
--cache_configs=cache_replacement/policy_learning/cache/configs/eviction_policy/belady.json \
--memtrace_file=/share/$GROUP/traces/astar_313B_test.csv
```



Section 3 :

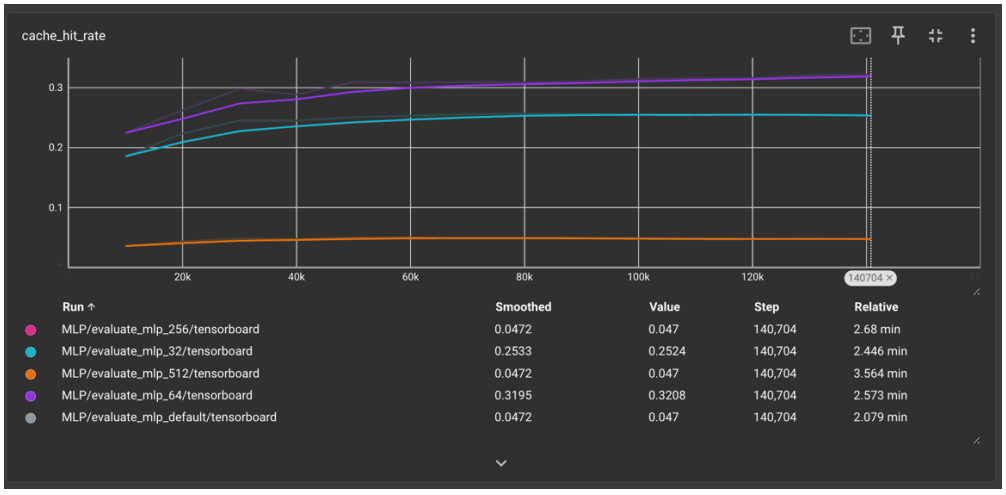
Task 1- Vary the size/width (number of neurons in a layer)

- No. of neurons – 32,64,128,256,512.

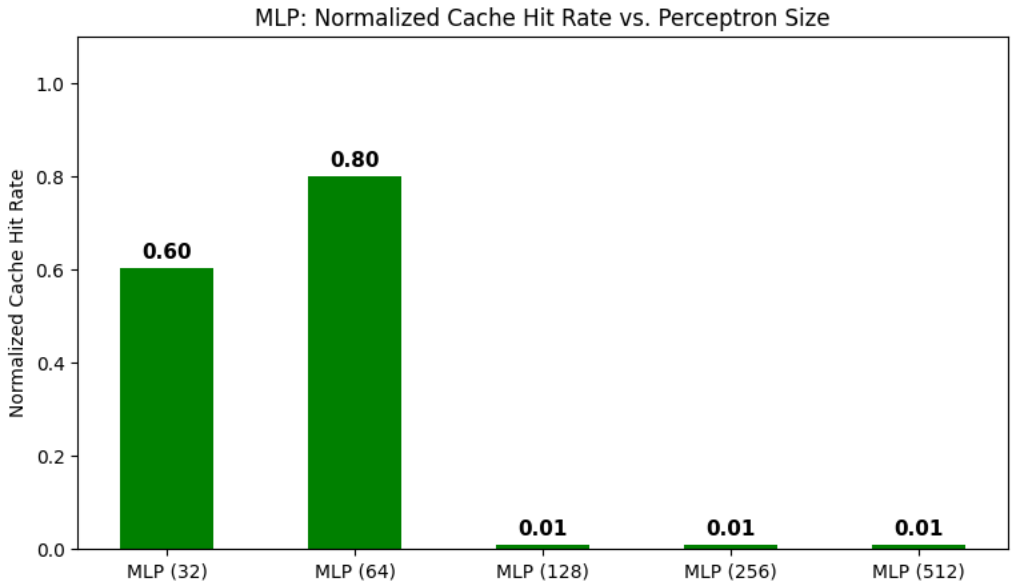
Evaluation Bash Script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -J evaluation_mlp_32
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/err.%J

source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP

python3 -m cache_replacement.policy_learning.cache.main \
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments \
  --experiment_name=evaluate_mlp_32 \
  --cache_configs="cache_replacement/policy_learning/cache/configs/default.json" \
  --cache_configs="cache_replacement/policy_learning/cache/configs/eviction_policy/learned.json" \
  --memtrace_file="/share/$GROUP/traces/astar_313B_test.csv" \
  --config_bindings="associativity=16" \
  --config_bindings="capacity=2097152" \
  --config_bindings="eviction_policy.scorer.checkpoint=\"/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments/train_astar_MLP_32/checkpoints/20000.ckpt\"" \
  --config_bindings="eviction_policy.scorer.config_path=\"/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments/train_astar_MLP_32/model_config.json\"" \
  --warmup_period=0
```

Output Tensorboard:



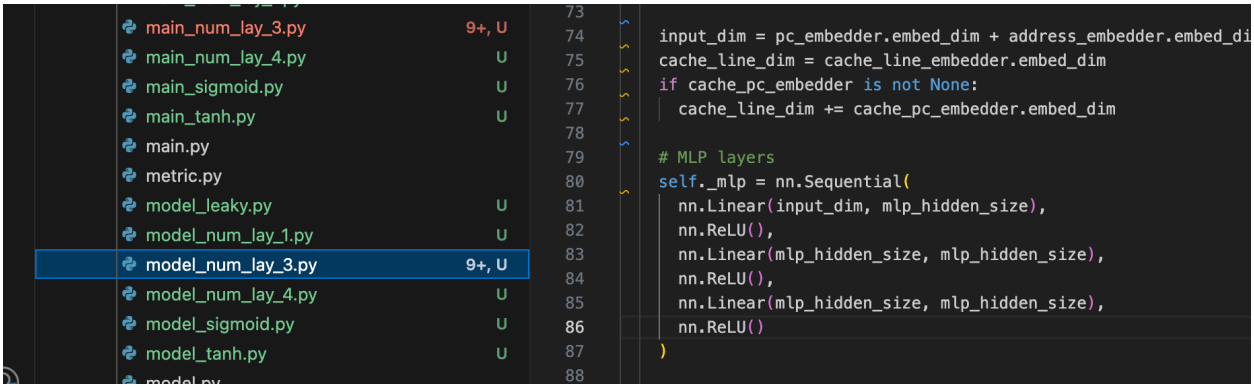
Run	Smoothed	Value	Step	Relative
MLP/evaluate_mlp_256/tensorboard	0.0472	0.047	140,704	2.68 min
MLP/evaluate_mlp_32/tensorboard	0.2533	0.2524	140,704	2.446 min
MLP/evaluate_mlp_512/tensorboard	0.0472	0.047	140,704	3.564 min
MLP/evaluate_mlp_64/tensorboard	0.3195	0.3208	140,704	2.573 min
MLP/evaluate_mlp_default/tensorboard	0.0472	0.047	140,704	2.079 min



As we can see, a perceptron with 32 or 64 neurons performs with a normalized cache hit rate of around 0.6 and 0.8 which is higher. And, when we increase the number of neurons, the results do not improve. This can be because of the case of overfitting, where making the model more complex doesn't actually help with improving the cache hit rate.

Taks 2 – Vary number of layers - 1,2,3,4

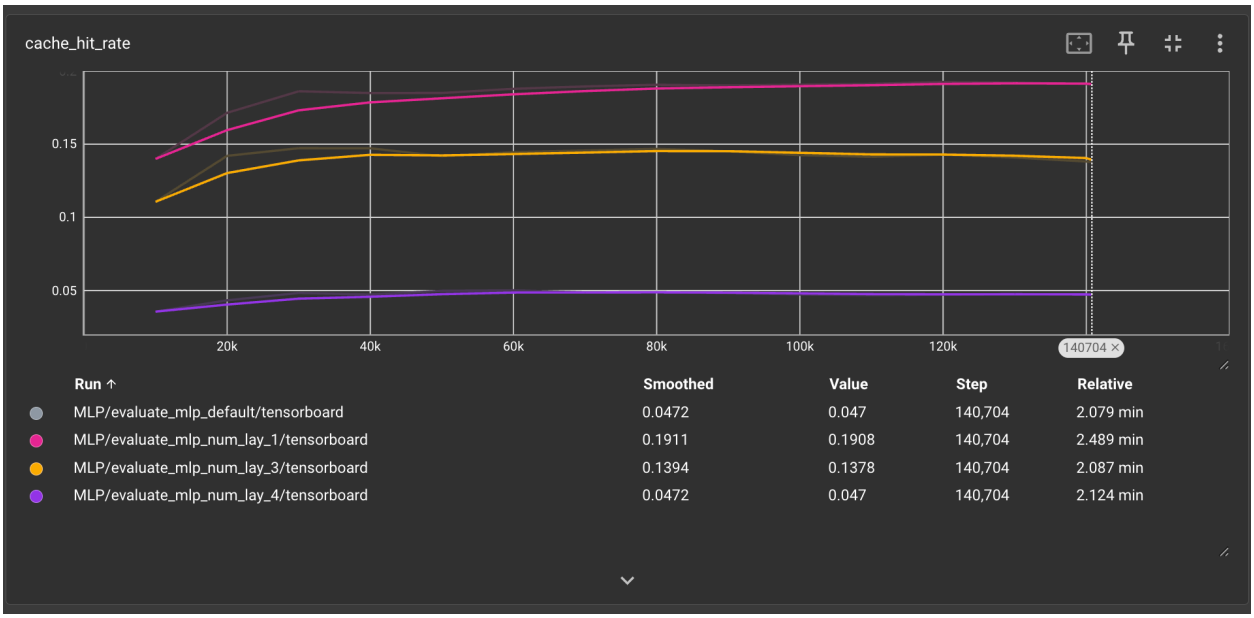
Other tasks code can be referred in the GH repo.

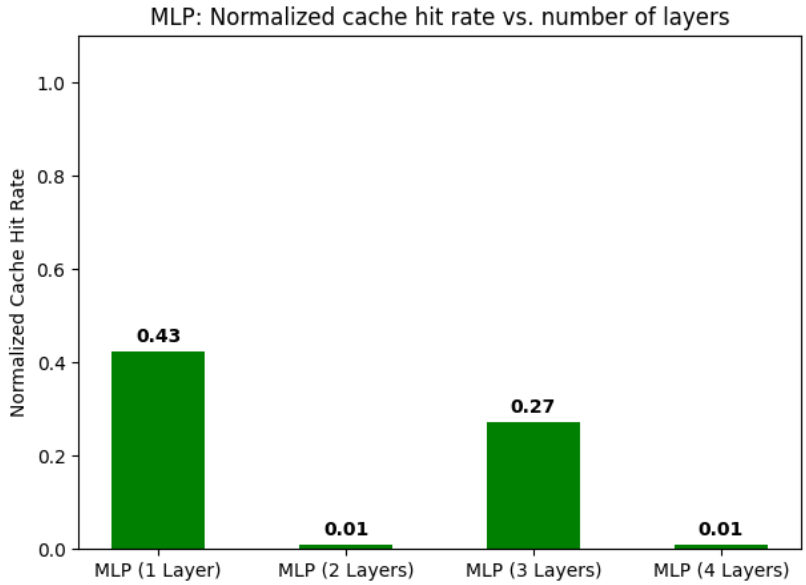


Bash Script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -q gpu
#BSUB -gpu "num=1"
#BSUB -J MLP_num_lay_1
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/err.%J


source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP

python3 -m cache_replacement.policy_learning.cache_model.main_num_lay_1 \
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments \
  --experiment_name=train_astar_MLP_num_lay_1 \
  --cache_configs=cache_replacement/policy_learning/cache/configs/default.json \
  --model_bindings="address_embedder.max_vocab_size=5000" \
  --train_memtrace=/share/$GROUP/traces/astar_313B_train.csv \
  --valid_memtrace=/share/$GROUP/traces/astar_313B_valid.csv
```

Tensorboard:





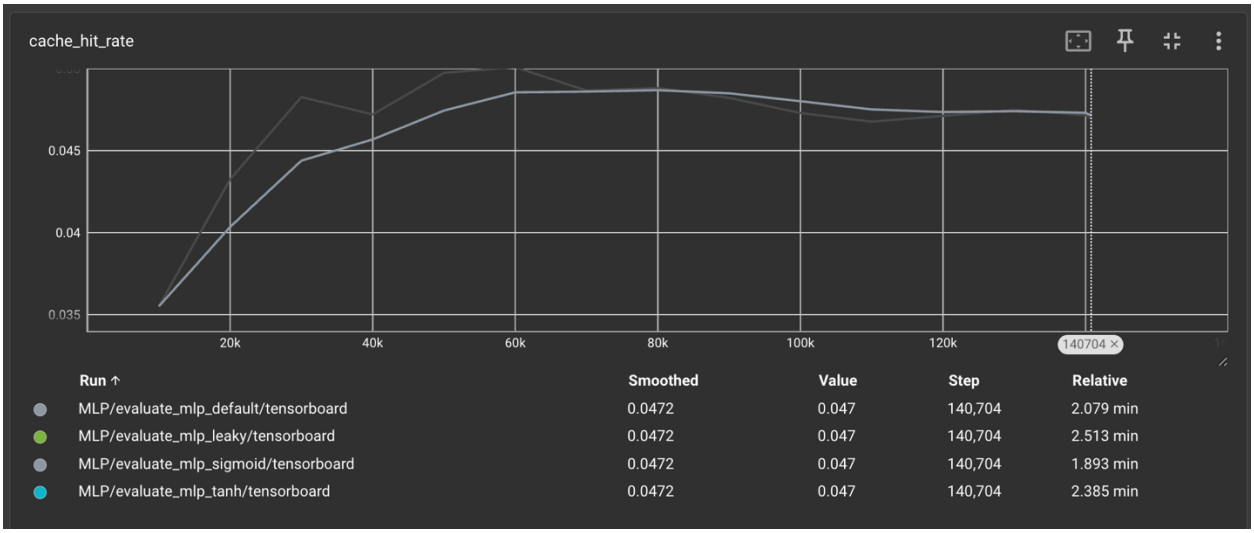
As we can see, the MLP with 1 layer achieves the highest normalized cache hit rate, around 0.43. However, when we move to an MLP with 2 layers, there's a sharp drop in the hit rate, about 0.01, which suggests there might be inefficiencies due to the added complexity. The MLP with 3 layers recovers a bit(could be because it extracted more meaningful interaction), reaching a hit rate of around 0.27. But then, the MLP with 4 layers drops back down to a very low hit rate, close to 0.01 again.

Task 3: activation functions - ReLU, Leaky ReLU, sigmoid, tanh

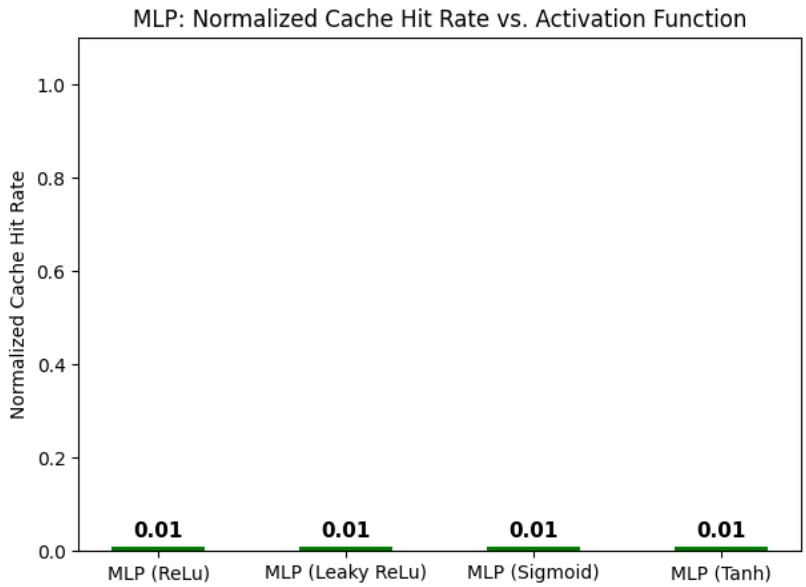
Bash Script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -q gpu
#BSUB -gpu "num=1"
#BSUB -J mlp_leaky
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/err.%J


source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP

python3 -m cache_replacement.policy_learning.cache_model.main_leaky\
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments \
  --experiment_name=train_astar_mlp_leaky \
  --cache_configs=cache_replacement/policy_learning/cache/configs/default.json \
  --model_bindings="address_embedder.max_vocab_size=5000" \
  --train_memtrace=/share/$GROUP/traces/astar_313B_train.csv \
  --valid_memtrace=/share/$GROUP/traces/astar_313B_valid.csv
```

Tensorboard:



Run ↑	Smoothed	Value	Step	Relative
MLP/evaluate_mlp_default/tensorboard	0.0472	0.047	140,704	2.079 min
MLP/evaluate_mlp_leaky/tensorboard	0.0472	0.047	140,704	2.513 min
MLP/evaluate_mlp_sigmoid/tensorboard	0.0472	0.047	140,704	1.893 min
MLP/evaluate_mlp_tanh/tensorboard	0.0472	0.047	140,704	2.385 min



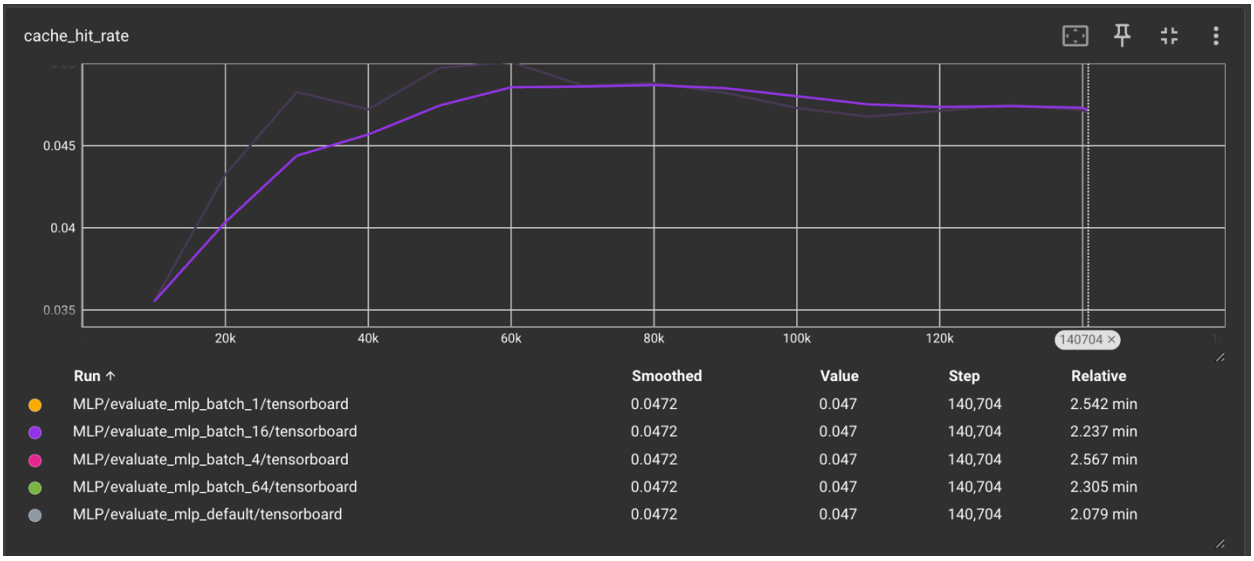
All activation functions perform similarly, showing no difference in the normalized cache hit rate.

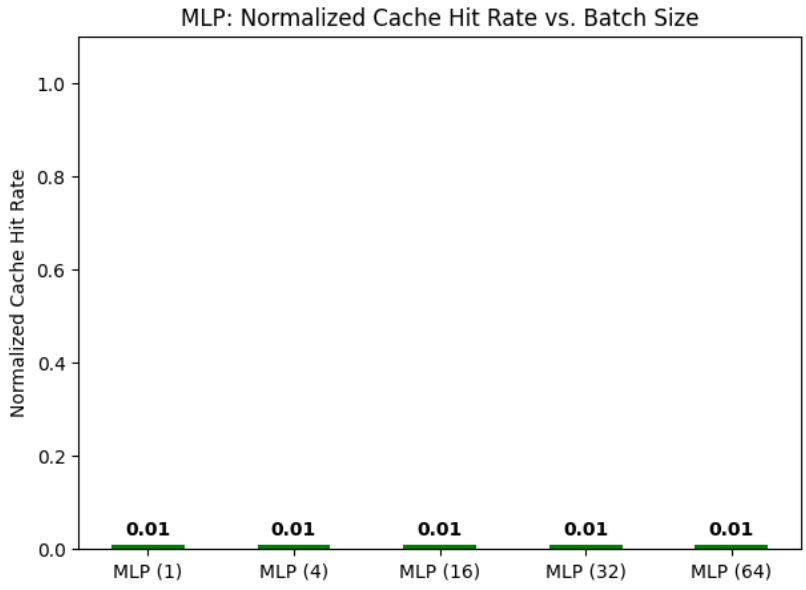
Task 4: batch sizes = 1(incremental), 4, 16, 32 (default), 64.

Bash Script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -q gpu
#BSUB -gpu "num=1"
#BSUB -J mlp_batch_1
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/err.%J


source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP

python3 -m cache_replacement.policy_learning.cache_model.main\
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments \
  --experiment_name=train_astar_mlp_batch_1 \
  --batch_size=1 \
  --cache_configs=cache_replacement/policy_learning/cache/configs/default.json \
  --model_bindings="address_embedder.max_vocab_size=5000" \
  --train_memtrace=/share/$GROUP/traces/astar_313B_train.csv \
  --valid_memtrace=/share/$GROUP/traces/astar_313B_valid.csv
```

Tensorboard:





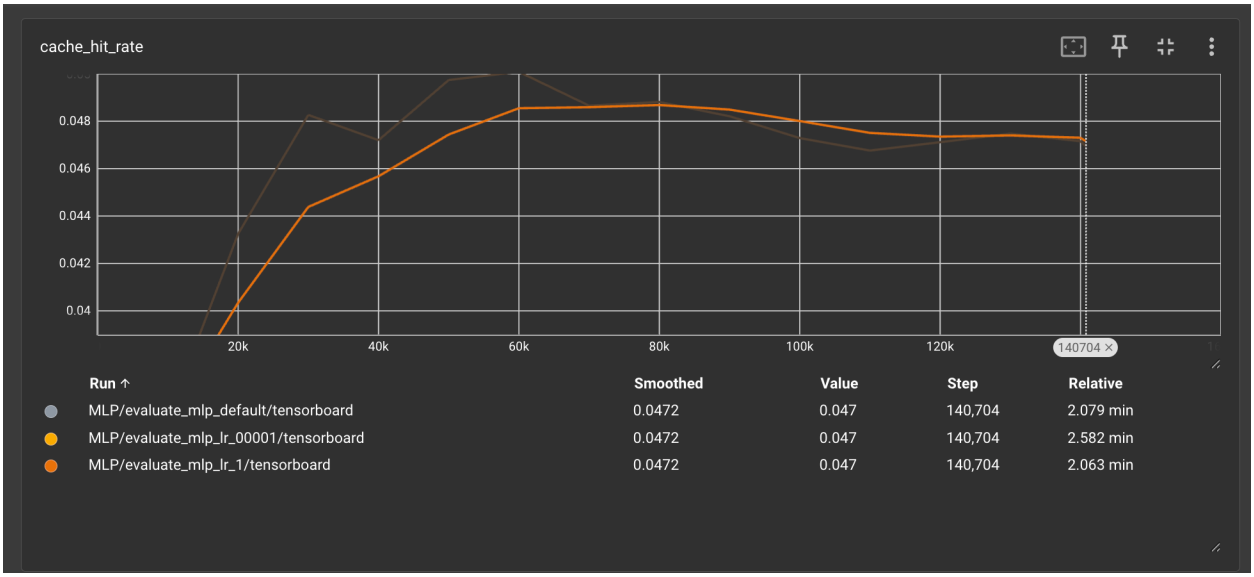
Similar performance is seen across all studies for different batch sizes.

Task 5: learning rates = 0.00001, 0.001(default), 0.1

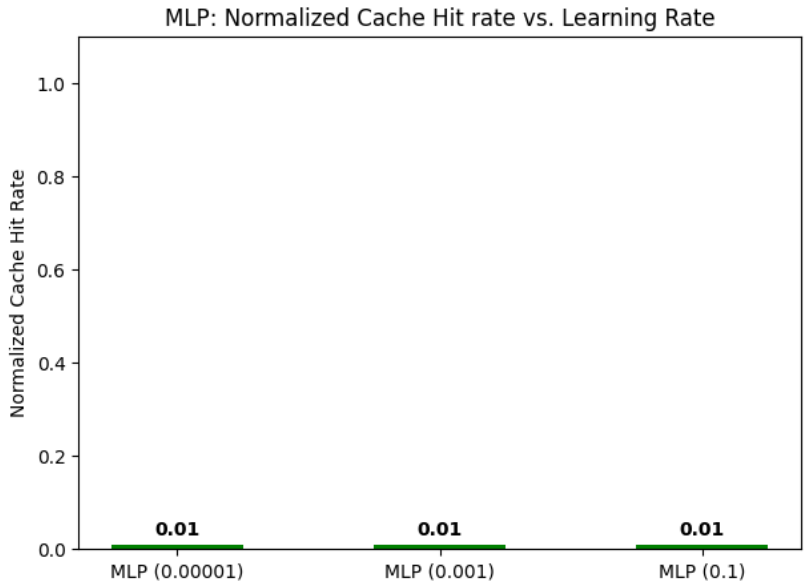
Bash Script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -q gpu
#BSUB -gpu "num=1"
#BSUB -J mlp_lr_1
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/MLP/logs/err.%J


source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP

python3 -m cache_replacement.policy_learning.cache_model.main\
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/MLP/experiments \
  --experiment_name=train_astar_mlp_lr_1 \
  --cache_configs=cache_replacement/policy_learning/cache/configs/default.json \
  --model_bindings="address_embedder.max_vocab_size=5000" \
  --model_bindings="lr=0.1" \
  --train_memtrace=/share/$GROUP/traces/astar_313B_train.csv \
  --valid_memtrace=/share/$GROUP/traces/astar_313B_valid.csv
```

Tensorboard:



Run ↑	Smoothed	Value	Step	Relative
MLP/evaluate_mlp_default/tensorboard	0.0472	0.047	140,704	2.079 min
MLP/evaluate_mlp_lr_00001/tensorboard	0.0472	0.047	140,704	2.582 min
MLP/evaluate_mlp_lr_1/tensorboard	0.0472	0.047	140,704	2.063 min



As we can see, the learning rates also have no impact on the normalized cache hit rate, resulting in a value of 0.01.

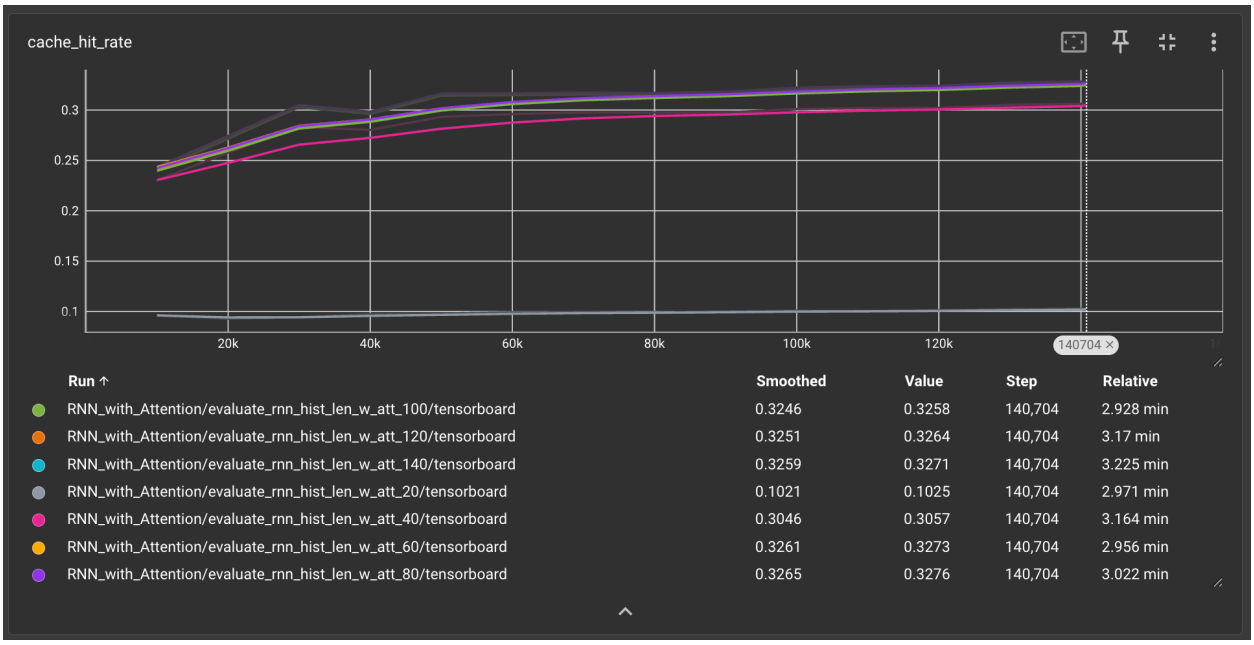
Section 4:

Task 1: RNN With Attention- history lengths - 20, 40, 60, 80, 100, 120, 140

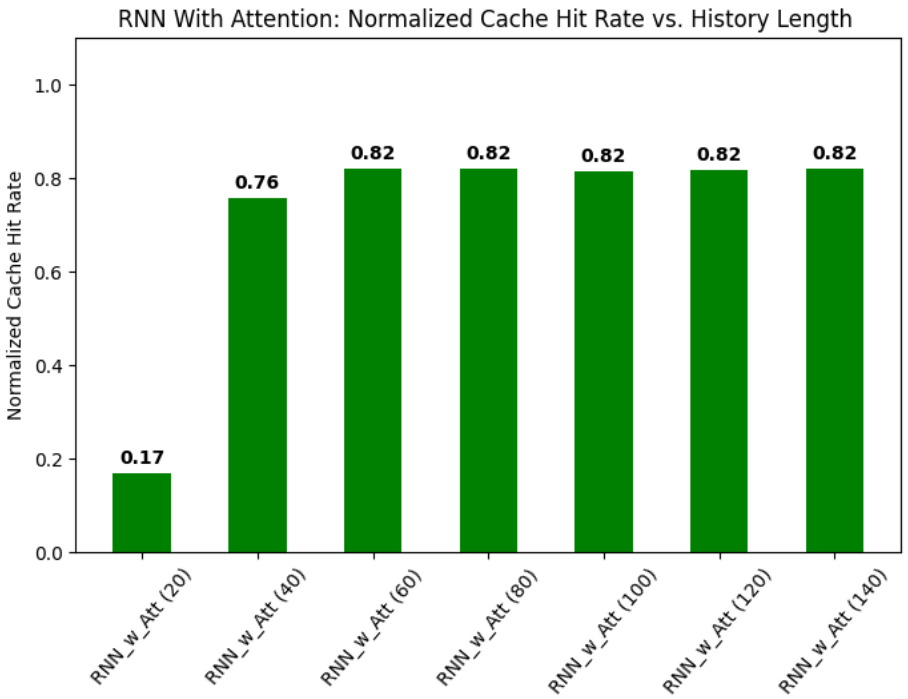
Bash Script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -J evaluation_rnn_hist_len_w_att_100
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/RNN_with_Attention/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/RNN_with_Attention/logs/err.%J

source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/RNN_with_Attention

python3 -m cache_replacement.policy_learning.cache.main \
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/RNN_with_Attention/experiments \
  --experiment_name=evaluate_rnn_hist_len_w_att_100\
  --cache_configs="cache_replacement/policy_learning/cache/configs/default.json" \
  --cache_configs="cache_replacement/policy_learning/cache/configs/eviction_policy/learned.json" \
  --memtrace_file="/share/$GROUP/traces/astar_313B_test.csv" \
  --config_bindings="associativity=16" \
  --config_bindings="capacity=2097152" \
  --config_bindings="eviction_policy.scorer.checkpoint=\"/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-
1/models/RNN_with_Attention/experiments/rnn_hist_len_w_att_100/checkpoints/20000.ckpt\"" \
  --config_bindings="eviction_policy.scorer.config_path=\"/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-
1/models/RNN_with_Attention/experiments/rnn_hist_len_w_att_100/model_config.json\"" \
  --warmup_period=0
```

Tensorboard:



Run	Smoothed	Value	Step	Relative
RNN_with_Attention/evaluate_rnn_hist_len_w_att_100/tensorboard	0.3246	0.3258	140,704	2.928 min
RNN_with_Attention/evaluate_rnn_hist_len_w_att_120/tensorboard	0.3251	0.3264	140,704	3.17 min
RNN_with_Attention/evaluate_rnn_hist_len_w_att_140/tensorboard	0.3259	0.3271	140,704	3.225 min
RNN_with_Attention/evaluate_rnn_hist_len_w_att_20/tensorboard	0.1021	0.1025	140,704	2.971 min
RNN_with_Attention/evaluate_rnn_hist_len_w_att_40/tensorboard	0.3046	0.3057	140,704	3.164 min
RNN_with_Attention/evaluate_rnn_hist_len_w_att_60/tensorboard	0.3261	0.3273	140,704	2.956 min
RNN_with_Attention/evaluate_rnn_hist_len_w_att_80/tensorboard	0.3265	0.3276	140,704	3.022 min



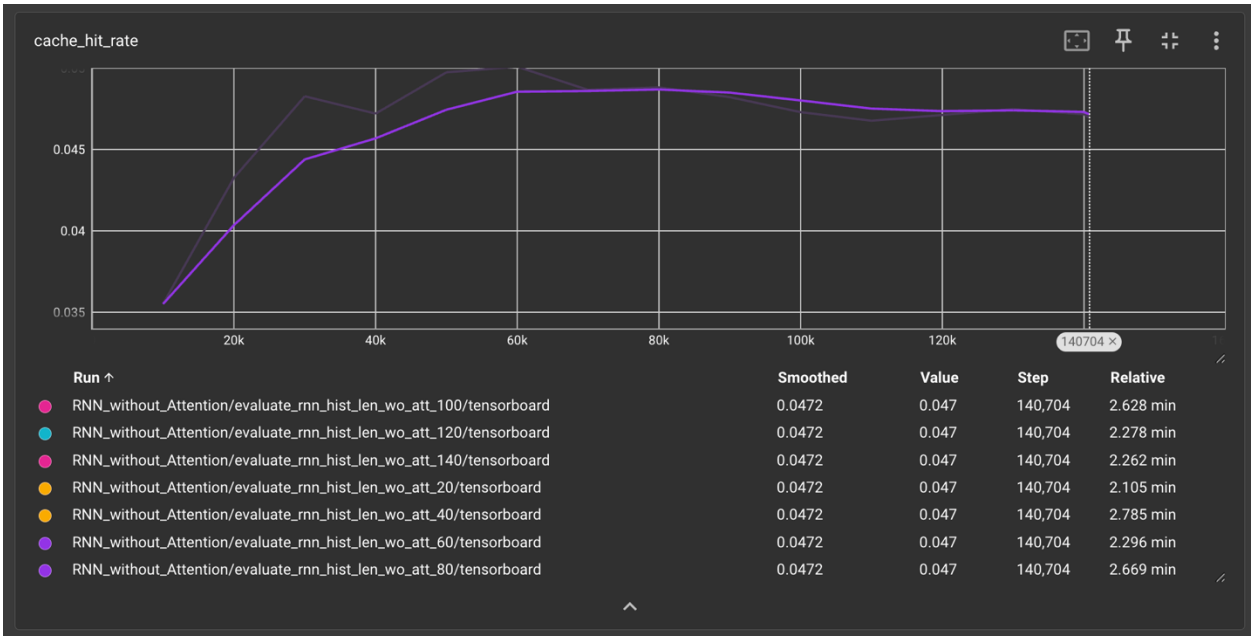
It looks like the RNN with attention gets better as we increase the history length. With a shorter history of 20, the hit rate is quite low at around 0.17, but when we extend it to 40, there's a big jump to about 0.76. After that, from 60 to 140, the performance levels off at 0.82. This shows that giving the model more memory helps up to a point, but after a certain limit, adding more history doesn't make a difference.

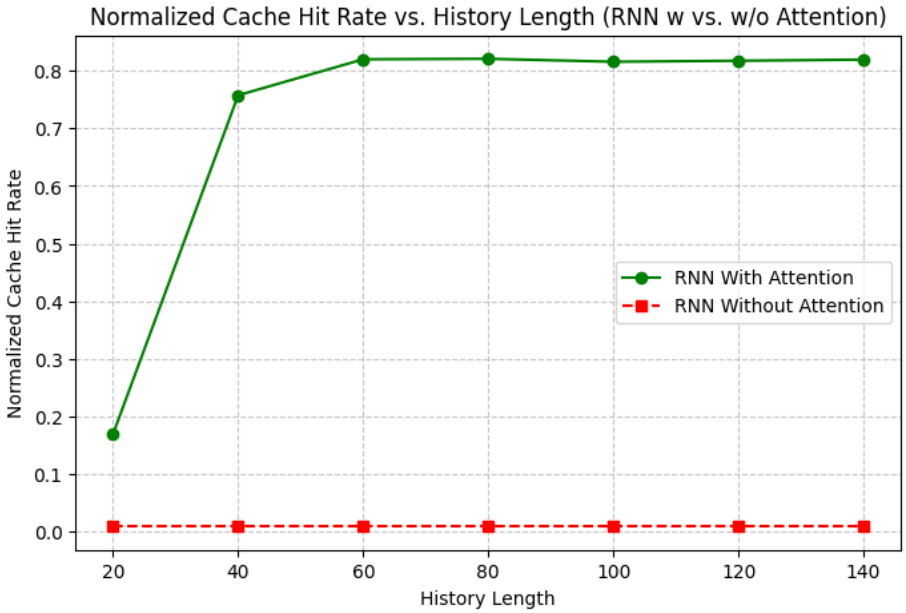
Task 2 : RNN Without Attention- history lengths - 20, 40, 60, 80, 100, 120, 140

Bash Script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -J evaluation_rnn_hist_len_wo_att_100
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/RNN_without_Attention/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/RNN_without_Attention/logs/err.%J

source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/RNN_without_Attention

python3 -m cache_replacement.policy_learning.cache.main \
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/RNN_without_Attention/experiments \
  --experiment_name=evaluate_rnn_hist_len_wo_att_100\
  --cache_configs="cache_replacement/policy_learning/cache/configs/default.json" \
  --cache_configs="cache_replacement/policy_learning/cache/configs/eviction_policy/learned.json" \
  --memtrace_file="/share/$GROUP/traces/astar_313B_test.csv" \
  --config_bindings="associativity=16" \
  --config_bindings="capacity=2097152" \
  --config_bindings="eviction_policy.scorer.checkpoint=\"/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-
1/models/RNN_without_Attention/experiments/rnn_hist_len_wo_att_100/checkpoints/20000.ckpt\"" \
  --config_bindings="eviction_policy.scorer.config_path=\"/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-
1/models/RNN_without_Attention/experiments/rnn_hist_len_wo_att_100/model_config.json\"" \
  --warmup_period=0
```

Tensorboard:



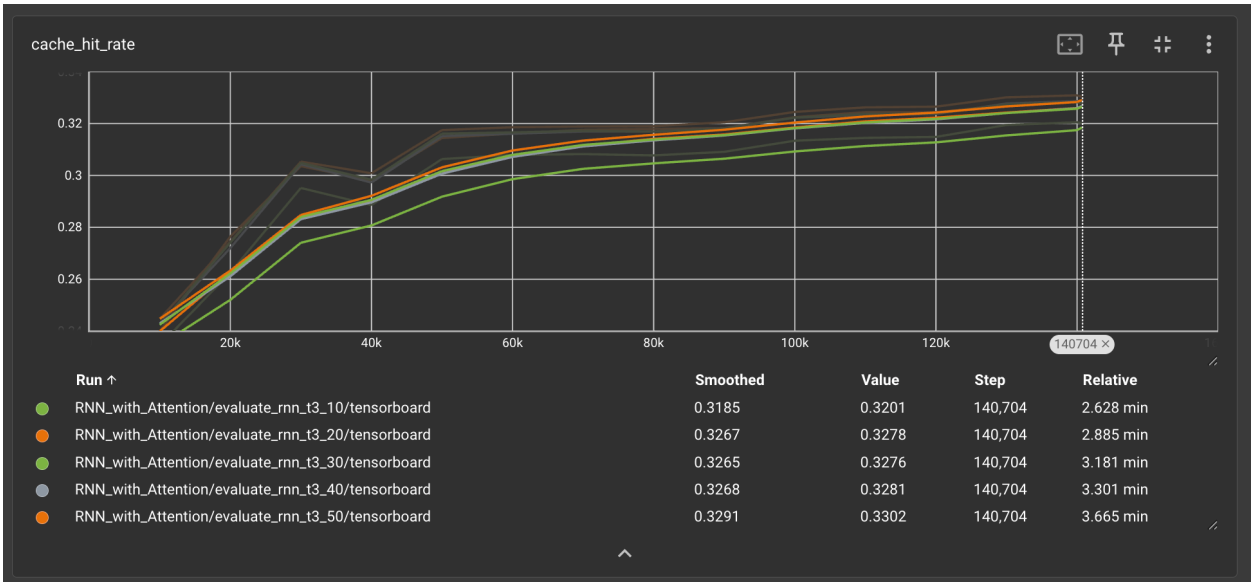


At all given history length, RNN with Attention outperforms without attention models, which indicates the importance of attention.

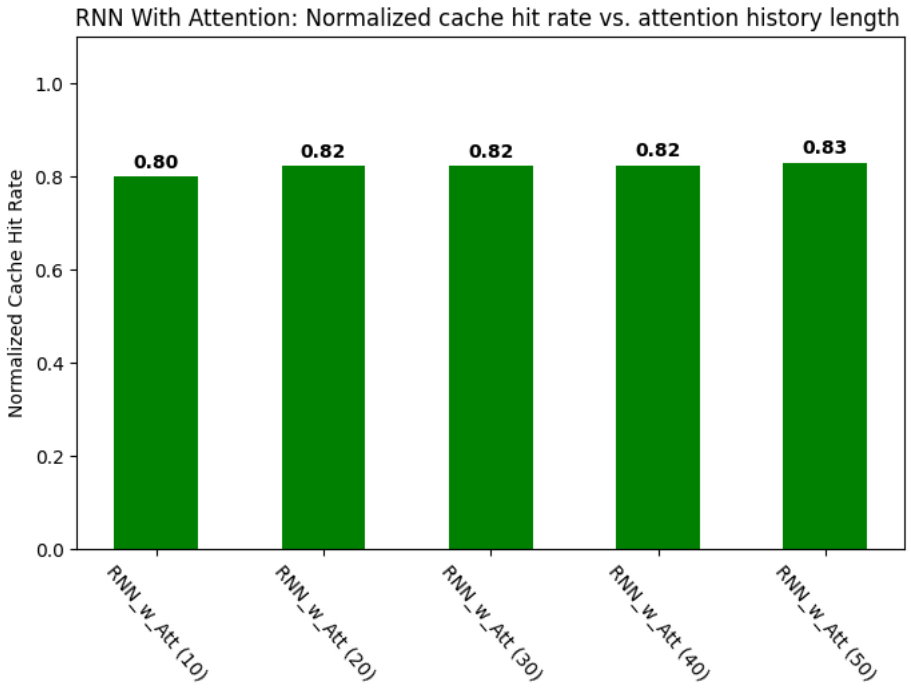
Task 3: attention history lengths - 10,20,30,40,50.
Bash Script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -q gpu
#BSUB -gpu "num=1"
#BSUB -J rnn_t3_10
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/RNN_with_Attention/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/RNN_with_Attention/logs/err.%J


source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/RNN_with_Attention

python3 -m cache_replacement.policy_learning.cache_model.main \
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/RNN_with_Attention/experiments \
  --experiment_name=rnn_t3_10 \
  --cache_configs=cache_replacement/policy_learning/cache/configs/default.json \
  --model_bindings="loss=[\"ndcg\", \"reuse_dist\"]" \
  --model_bindings="max_attention_history=10" \
  --model_bindings="address_embedder.max_vocab_size=5000" \
  --train_memtrace=/share/$GROUP/traces/astar_313B_train.csv \
  --valid_memtrace=/share/$GROUP/traces/astar_313B_valid.csv
```

Tensorboard:



Run	Smoothed	Value	Step	Relative
RNN_with_Attention/evaluate_rnn_t3_10/tensorboard	0.3185	0.3201	140,704	2.628 min
RNN_with_Attention/evaluate_rnn_t3_20/tensorboard	0.3267	0.3278	140,704	2.885 min
RNN_with_Attention/evaluate_rnn_t3_30/tensorboard	0.3265	0.3276	140,704	3.181 min
RNN_with_Attention/evaluate_rnn_t3_40/tensorboard	0.3268	0.3281	140,704	3.301 min
RNN_with_Attention/evaluate_rnn_t3_50/tensorboard	0.3291	0.3302	140,704	3.665 min



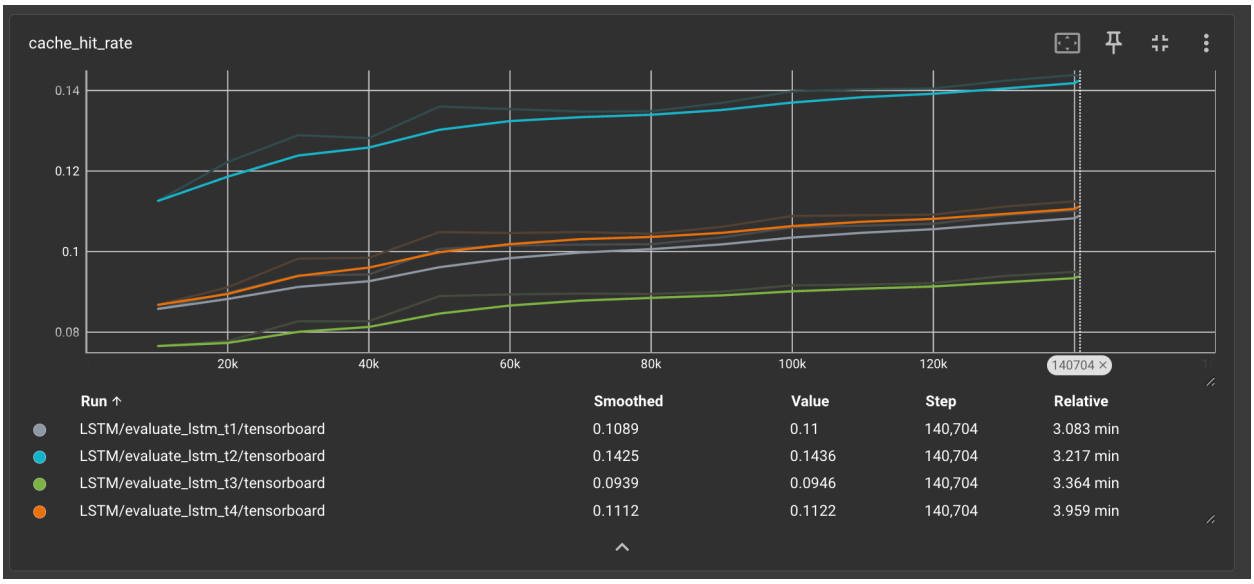
The RNN with attention performs almost the same across different attention lengths, ranging from 10 to 50. The best performance is at length 50 with a hit rate of 0.83, while the worst is at length 10 with 0.81. However, the difference is very small, meaning attention length doesn't have a major impact on performance.

Section 5: LSTM Task 1 to 4:

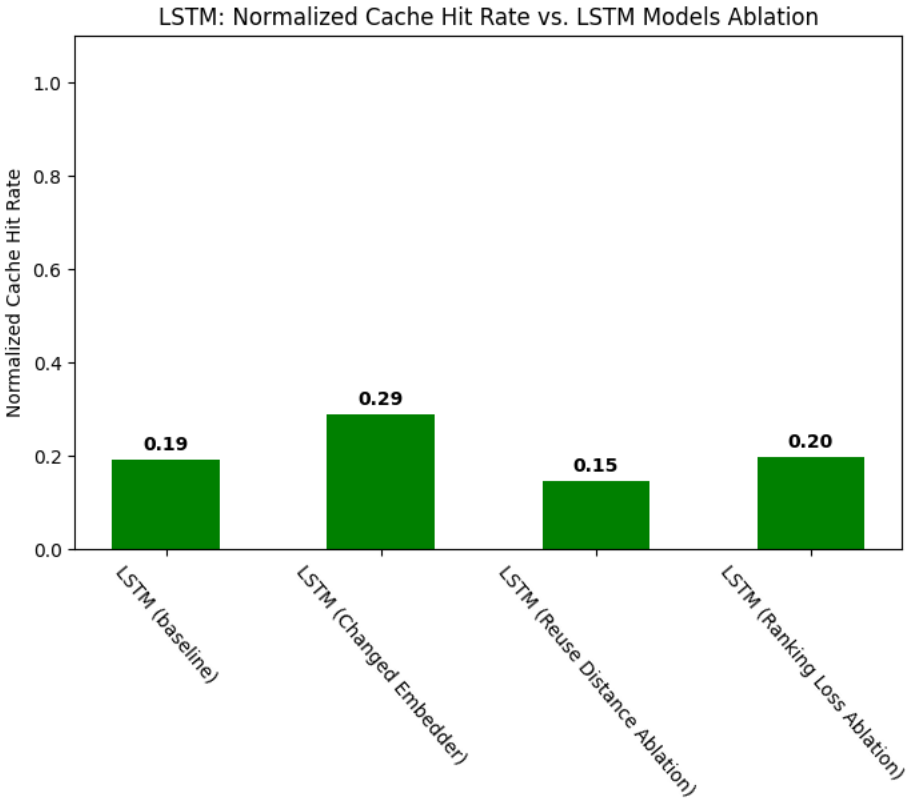
Example bash script:

```
#!/bin/bash
#BSUB -n 1
#BSUB -W 48:00
#BSUB -J evaluation_lstm_t2
#BSUB -o /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/LSTM/logs/out.%J
#BSUB -e /share/csc591s25/sgundew/GenAI-for-Systems-Gym/homework-1/models/LSTM/logs/err.%J

source ~/.bashrc
conda activate /share/$GROUP/conda_env/new_env
cd /share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/LSTM

python3 -m cache_replacement.policy_learning.cache.main \
  --experiment_base_dir=/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/LSTM/experiments \
  --experiment_name=evaluate_lstm_t2\
  --cache_configs="cache_replacement/policy_learning/cache/configs/default.json" \
  --cache_configs="cache_replacement/policy_learning/cache/configs/eviction_policy/learned.json" \
  --memtrace_file="/share/$GROUP/traces/astar_313B_test.csv" \
  --config_bindings="associativity=16" \
  --config_bindings="capacity=2097152" \
  --config_bindings="eviction_policy.scorer.checkpoint=\"/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/LSTM/experiments/LSTM_T2/checkpoints/20000.ckpt\"" \
  --config_bindings="eviction_policy.scorer.config_path=\"/share/$GROUP/$USER/GenAI-for-Systems-Gym/homework-1/models/LSTM/experiments/LSTM_T2/model_config.json\"" \
  --warmup_period=0
```

Tensorboard:



Run ↑	Smoothed	Value	Step	Relative
LSTM/evaluate_lstm_t1/tensorboard	0.1089	0.11	140,704	3.083 min
LSTM/evaluate_lstm_t2/tensorboard	0.1425	0.1436	140,704	3.217 min
LSTM/evaluate_lstm_t3/tensorboard	0.0939	0.0946	140,704	3.364 min
LSTM/evaluate_lstm_t4/tensorboard	0.1112	0.1122	140,704	3.959 min



The LSTM baseline gives a normalized cache hit rate of 0.19. When we changed the embedder, the hit rate increased to 0.29, suggesting that how embeddings are represented plays a key role in learning effective cache replacement patterns. Removing reuse distance caused a slight drop in performance compared to the baseline. Meanwhile, removing ranking loss led to a hit rate of 0.20 which is almost similar to baseline, and reuse distance optimization significantly impacts cache replacement, though the difference is relatively small..