Self-Optimizing Memory Controllers: A Reinforcement Learning Approach - ISCA 2008
Engin İpek¹,², Onur Mutlu², José F. Martínez¹, Rich Caruana¹
¹ Cornell University
² Microsoft Research

Presented by Marco Zeller
Outline

1. Motivation and Background
2. Mechanisms
3. Results
4. Summary
5. Discussion
Moore’s Law and Memory Controller

Projection according to ITRS 2007 Executive Summary
Off-Chip Memory Bandwidth Observations

Workload is increasing faster than the available bandwidth
Off-Chip Memory Bandwidth Observations

Workload is increasing faster than the available bandwidth

Higher pressure on off-chip interface with each new technology generation
Off-Chip Memory Bandwidth Observations

Workload is increasing faster than the available bandwidth

Higher pressure on off-chip interface with each new technology generation

Important to utilize available memory bandwidth efficiently
The Memory Controller
The Memory Controller

Diagram showing the memory architecture with cores and controllers.
Memory Scheduling

We consider 4 DRAM-Interface commands:
Memory Scheduling

We consider 4 DRAM-Interface commands:

1. Activate (row)
2. Precharge (row)
3. Read
4. Write

The paper distinguishes 5 commands (performance)
Memory Scheduling

We consider 4 DRAM-Interface commands:

1. Activate (row) ’Open’
2. Precharge (row) ’Close’
3. Read
4. Write

The paper distinguishes 5 commands (performance)
Row Buffer

Activate
Row Buffer

Activate
Row Buffer

Write
Row Buffer

Write
Row Buffer

Read
Row Buffer

Read

![Diagram of Row Buffer](image)
Row Buffer

Precharge
Row Buffer

Precharge
State of the Art Memory Controller

**FR-FCFS** (first-ready first-come first-serve) policy
Provides the best average performance

**Drawbacks:**
- Designed for average-case application behavior
- Does not consider long-term performance impacts
- Does not adapt its scheduling policy
State of the Art Memory Controller

**FR-FCFS** (first-ready first-come first-serve) policy
Provides the best average performance

**Drawbacks:**
- Designed for average-case application behavior
- Does not consider long-term performance impacts
- Does not adapt its scheduling policy
Assume the following workload: action (bank, row, column)

1. Read (0,0,0)  
2. Read (0,1,0)  
3. Write (0,1,0)  
4. Read (0,1,0)  
5. Read (0,0,1)  
6. Read (0,1,1)  
7. Read (1,0,0)  
8. Write (1,0,0)  
9. Read (1,0,0)  
10. Read (1,1,0)  
11. Read (1,0,1)  
12. Read (1,1,1)  

Where:
Activate and Precharge occupy 3 DRAM-cycles
Read and Write occupy 1 DRAM-cycles
60 cycles!
23 cycles!
22 cycles!
How Much Room Is There for Improvement?

Used for paper: "Optimistic Scheduler"
How Much Room Is There for Improvement?

Used for paper: "Optimistic Scheduler"

All timing constraints lifted (except DRAM data bus conflicts)
How Much Room Is There for Improvement?

Used for paper: "Optimistic Scheduler"

All timing constraints lifted (except DRAM data bus conflicts)

Able to use the full bandwidth at all time
How Much Room Is There for Improvement?

Used for paper: ”Optimistic Scheduler”

All timing constraints lifted (except DRAM data bus conflicts)

Able to use the full bandwidth at all time

Not realizable!

but easy to implement

useful approximation for upper bound
FR-FCFS Compared to Optimistic Scheduler
Novelty: Self-Optimizing Memory Controller

Observation:
Efficient memory controller must be able to:

1. do long time planning
2. learn
Novelty: Self-Optimizing Memory Controller

Observation:
Efficient memory controller must be able to:

1. do long time planning
2. learn

Solution:
Apply machine learning to solve a system architecture problem
Mechanisms

"Naturally formulated as an infinite-horizon discounted Markov Decision Process"
Potentially Complicated System

Retrieved from https://www.simplypsychology.org/operant-conditioning.html
Simple System for the Learning Agent
Reinforcement Learning Schematic

Environment

Agent

Action $a(t+1)$

Reward $r(t)$

State $s(t)$
Reinforcement Learning Schematic

Environment

Agent

System

DRAM Scheduler

Action $a(t+1)$

State $s(t)$

Reward $r(t)$

Scheduled Command $(t+1)$

State Attributes $(t)$

Data Bus Utilization $(t)$
How to Obtain a Good Set of State Attributes

More state attributes $\Rightarrow$ increased hardware complexity
How to Obtain a Good Set of State Attributes

More state attributes $\Rightarrow$ increased hardware complexity

Handpick potential state attributes (requires some intuition)
in the paper 226 candidates were identified
How to Obtain a Good Set of State Attributes

More state attributes ⇒ increased hardware complexity

Handpick potential state attributes (requires some intuition) in the paper 226 candidates were identified

Apply (linear) feature selection on those candidates
(Linear) Feature Selection

Automated process for finding a good (the best?) subset from N candidates
(Linear) Feature Selection

Automated process for finding a good (the best?) subset from N candidates

1. For every candidate (N) simulate an RL memory controller
(Linear) Feature Selection

Automated process for finding a good (the best?) subset from N candidates

1. For every candidate (N) simulate an RL memory controller
2. Select the attribute yielding best performance
(Linear) Feature Selection

Automated process for finding a good (the best?) subset from N candidates

1. For every candidate (N) simulate an RL memory controller
2. Select the attribute yielding best performance
3. Simulate for every not selected candidate (N-1) an RL memory controller using the not selected attribute and the selected one
(Linear) Feature Selection

Automated process for finding a good (the best?) subset from N candidates

1. For every candidate (N) simulate an RL memory controller
2. Select the attribute yielding best performance
3. Simulate for every not selected candidate (N-1) an RL memory controller using the not selected attribute and the selected one
4. Repeat until preferred number of attributes reached
State Attributes (simplified)

1. Number of reads (load/store misses)
2. Number of writes (write-backs)
3. Number of reads (load misses)
4. Order of a load by core relative to other loads by the same core
5. Number of writes in the transaction queue waiting for the row
6. Number of load misses waiting for the row
Reward Function

The agent optimizes for the sum of its future rewards
Reward Function

The agent optimizes for the sum of its future rewards

- Immediate reward of +1 every time it schedules a Read or Write
- No reward at all other times
Reward Function

The agent optimizes for the sum of its future rewards

- Immediate reward of +1 every time it schedules a Read or Write
- No reward at all other times

Problem: agent’s lifetime practically infinity
⇒ sum of the rewards will be infinity
Solution: Discounted Rewards

Discount parameter: $0 \leq \gamma < 1$

$$\mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i \cdot r_{t+i}\right] = \mathbb{E}[r_t + \gamma^1 \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \gamma^3 \cdot r_{t+3} + ...]$$

Store these for all state-action pairs
⇒ Q-values
Example: Infinite-Horizon Discounted Markov Decision Process
Key Results
Experimental Setup

Loosely based on Intel’s Nehalem processor
4 GHz Quadcore Processor
Shared L2 Cache: 4MB

Main Memory: DDR2 6.4 GB/s (1, 2, 4 Channels)
Transaction Queue: 64 Entries

Benchmarks: 9 mostly scientific memory-intensive applications from \textit{Data Mining}, \textit{NAS OpenMP}, \textit{SPEC OpenMP}, \textit{Splash-2}

Simulator: heavily modified SESC
Speedup Compared to FR-FCFS

(27% of the possible speedup)

Marco Zeller 2018-10-24 27
Speedup Compared to FR-FCFS

5%-33% speedup

19% speedup on average

"(27% of the possible speedup)"
Data-Bus Utilization
Fixed Policy With Same State Information

RL uses more information about the system than FR-FCFS
Fixed Policy With Same State Information

RL uses more information about the system than FR-FCFS

Generate a family of policies based on additional state-attributes
Fixed Policy With Same State Information

RL uses more information about the system than FR-FCFS

Generate a family of policies based on additional state-attributes

Pick the one that performs best in simulations

⇒ Family-BEST
Fixed Policy With Same State Information

- Using more state-attributes to improve FR-FCFS yields only 5% speedup on average.
Fixed Policy With Same State Information

using more state-attributes to improve FR-FCFS yields only 5% speedup on average
Speedup Compared to a Static RL Policy

The diagram shows the speedup compared to a static RL policy using an offline RL memory controller. On average, this yields only 8% speedup. Therefore, online learning is essential.
Speedup Compared to a Static RL Policy

using an offline RL Memory Controller yields only 8% speedup on average

⇒ online learning is essential
More Than Two Cores?

Can the Q-values successfully converge in the presence of potential interactions between schedulers?
More Than Two Cores?

Can the Q-values successfully converge in the presence of potential interactions between schedulers? Yes!

![Graph showing speedup over FR-FCFS for different cores and channels]

- ART, CG, EQUAKE, FFT, MG, OCEAN, RADIX, SCALPARC, SWIM, G-MEAN, G-MEAN'
Speedup Compared to Two Memory Channels

Hardware overhead:

1. logic required to compute state attributes
2. logic required to estimate and update Q-values
3. SRAM arrays required to store Q-values.

8192 distinct Q-values (32bits each)

⇒ 32kB of on-chip storage
Speedup Compared to Two Memory Channels

How about **increasing the memory bandwidth** instead of deploying a RL Memory Controller?

![Graph showing speedup comparison](image)
Speedup Compared to Two Memory Channels

How about increasing the memory bandwidth instead of deploying a RL Memory Controller?

RL Memory Controller can deliver 50% of the performance increase of doubling the bandwidth

⇒ lower cost than an over-provisioned system
Summary

Problem: Off-chip memory bandwidth bottleneck

Goal: Improve usage of available bandwidth

Observation: Fixed policy memory controller are not optimal

Key idea: Apply RL to a computer architecture problem

self-optimizing memory controller

Key result: 19% average speedup compared to state-of-the-art controller (between 5% and 33% speedup for every tested application)

Scalability to more processors/memory-channels
Strengths and Weaknesses
Strengths

Improves the memory bandwidth usage and therefore the performance.
Strengths

Improves the memory bandwidth usage and therefore the performance.

Reduces the human design effort for the memory controller.

Black-box-model
Strengths

Improves the memory bandwidth usage and therefore the performance.

Reduces the human design effort for the memory controller. Black-box-model

Well written.
Weaknesses

State attributes attribute candidates identified *based on intuition*
Relatively high number (226) of candidates
Weaknesses

State attributes attribute candidates identified based on intuition
Relatively high number (226) of candidates

The (linear) feature selection process used does not take into account potentially important interactions between attributes
Weaknesses

State attributes attribute candidates identified based on intuition
Relatively high number (226) of candidates

The (linear) feature selection process used does not take into account potentially important interactions between attributes

The reward function proposed in the paper might not be ideal
⇒ Bias with all machine learning algorithms
Weaknesses

State attributes attribute candidates identified based on intuition
Relatively high number (226) of candidates

The (linear) feature selection process used does not take into account potentially important interactions between attributes

The reward function proposed in the paper might not be ideal
⇒ Bias with all machine learning algorithms

The reward function proposed in the paper does not easily generalize
Thoughts and Ideas
My Opinion about the Paper

It is worthwhile reading.
My Opinion about the Paper

It is worthwhile reading.

Although ten years old still relevant (more than ever)
My Opinion about the Paper

It is worthwhile reading.

Although ten years old still relevant (more than ever)

Interesting symbioses of two not often related topics (Machine Learning and Micro-architecture)
My Opinion about the Paper

It is worthwhile reading.

Although ten years old still relevant (more than ever)

Interesting symbioses of two not often related topics (Machine Learning and Micro-architecture)

Detailed description about possible implementation
My Opinion about the Paper

It is worthwhile reading.

Although ten years old still relevant (more than ever)

Interesting symbioses of two not often related topics (Machine Learning and Micro-architecture)

Detailed description about possible implementation

Suggested reading: MORSE Multi-objective Reconfigurable Self-optimizing Memory Scheduler - 2011
Takeaways
Combined knowledge from two little related fields

The authors took ideas from the field of Data-Processing and applied them to design a ‘superior’ Micro-Architecture.
Combined knowledge from two little related fields

The authors took ideas from the field of **Data-Processing** and applied them to design a 'superior' **Micro-Architecture**.

Remember this was before the Machine Learning Hype!
Combined knowledge from two little related fields

The authors took ideas from the field of **Data-Processing** and applied them to design a ’superior’ **Micro-Architecture**.

Remember this was before the Machine Learning Hype!

"To our knowledge, this paper is the first to propose such a scheduler along with a rigorous methodology to designing self-optimizing DRAM controllers."
What can we learn from this paper?

Thinking in a unconventional way can lead to good results

For that it is important to not only be an expert in one field, one does need to know about other fields to get inspiration

In addition to that, one needs the ability to carry out the ideas inspired by these other fields in order for them to form something more than a just dream
Discussion

Questions . . .
Ensuring Correct Operation 1

The scheduler’s decisions are restricted to picking among the set of legal commands each cycle.

Care must be taken to ensure that the system is guaranteed to make forward progress regardless of the scheduler’s decisions:

1. the scheduler is not permitted to select NOPs when other legal commands are available
2. the scheduler cannot choose to activate an arbitrary row with no pending requests
3. the scheduler is not allowed to precharge a newly activated row until it issues a read or write command to it.
Ensuring Correct Operation 2

Starvation

⇒ Timeout policy: any request that has been pending for a fixed (but large - in our case 10,000) number of cycles is completed in its entirety before other commands can be considered for scheduling.

DRAM refresh

we do not allow the RL controller to dictate the refresh schedule. Instead, at the end of a refresh interval, the RL scheduler is disabled, the appropriate rows are refreshed, and then control is returned to the RL scheduler.
Critique From the ATLAS Paper

"Other scheduling algorithms have been proposed to improve DRAM throughput in single-threaded, multi-threaded, or streaming systems. None of these works consider the existence of multiple competing threads sharing the memory controllers (as happens in a multi-core system)."

Link: ATLAS Paper
Bigger Picture: General issues with Memory

Key issues to tackle:

- Enable reliability at low cost/high capacity
- Reduce energy
- Reduce latency
- Improve bandwidth
- Reduce waste (capacity, bandwidth, latency)
- Enable computation close to data

according to: Memory Systems course, Technion, Summer 2018
Bigger Picture: General issues with Memory

Key issues to tackle:

- Enable reliability at low cost/high capacity
- Reduce energy
- Reduce latency
- Improve bandwidth
- Reduce waste (capacity, bandwidth, latency)
- Enable computation close to data

according to: Memory Systems course, Technion, Summer 2018
Difficulties in Optimizing Memory Controllers

1. The controller needs to obey all DRAM timing constraints to provide correct functionality

2. The controller must intelligently prioritize DRAM commands from different memory requests to optimize system performance.

Current memory controllers use relatively simple policies to schedule DRAM accesses.
Machine Learning and its Limits

Study of computer programs and algorithms that learn about their environment and improve automatically with experience.

TODO: PICTURE FUNCTION (on blackboard ?)
Applicability of RL to DRAM Scheduling

To apply RL to DRAM scheduling an agent is defined. The agent interacts with its environment over a discrete set of time steps. At each step, the agent senses the current state of its environment, and executes an action. This results in a change in the state of the environment (which the agent can sense in the next time step), and produces an immediate reward. The agent’s goal is to maximize its long-term cumulative reward by learning an optimal policy that maps states to actions.
Design Paradigms for the Agent

Temporal credit assignment  The agent needs to learn how to assign credit and blame to past actions for each observed immediate reward.

Exploration vs. exploitation  Too little exploration of the environment can cause the agent to commit to sub-optimal policies early on, whereas excessive exploration can result in long periods during which the agent executes sub-optimal actions to explore its environment.

Generalization  it is exceedingly improbable for the agent to experience the same state more than once over its lifetime.
State-Attributes (not simplified)

1. Number of reads (load/store misses) in the transaction queue.
2. Number of writes (writebacks) in the transaction queue.
3. Number of reads in the transaction queue that are load misses.
4. If the command is related to a load miss by core C in the transaction queue, the load’s order in C’s dynamic instruction stream relative to other loads by C with requests in the transaction queue.
5. Number of writes in the transaction queue waiting for the row referenced by the command under consideration.
6. Number of load misses in the transaction queue waiting for the row referenced by the command under consideration which have the oldest sequence number among all load misses in the transaction queue from their respective cores.
## Experimental Setup

<table>
<thead>
<tr>
<th>Processor Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>4.0 GHz</td>
</tr>
<tr>
<td>Number of cores</td>
<td>4</td>
</tr>
<tr>
<td>Number of SMT Contexts</td>
<td>2 per core</td>
</tr>
<tr>
<td>Fetch/issue/commit width</td>
<td>4/4/4</td>
</tr>
<tr>
<td>Int/FP/Ld/St/Br Units</td>
<td>2/2/2/2/2/2</td>
</tr>
<tr>
<td>Int/FP Multipliers</td>
<td>1/1</td>
</tr>
<tr>
<td>Int/FP issue queue size</td>
<td>32/32 entries</td>
</tr>
<tr>
<td>ROB (reorder buffer) entries</td>
<td>96</td>
</tr>
<tr>
<td>Int/FP registers</td>
<td>96 / 96</td>
</tr>
<tr>
<td>Ld/St queue entries</td>
<td>24/24</td>
</tr>
<tr>
<td>Max. unresolved br.</td>
<td>24</td>
</tr>
<tr>
<td>Br. mispred. penalty</td>
<td>9 cycles min.</td>
</tr>
<tr>
<td>Br. predictor</td>
<td>Alpha 21264 (tournament)</td>
</tr>
<tr>
<td>RAS entries</td>
<td>32</td>
</tr>
<tr>
<td>BTB size</td>
<td>512 entries, direct-mapped</td>
</tr>
<tr>
<td>iL1/dL1 size</td>
<td>32 kB</td>
</tr>
<tr>
<td>iL1/dL1 block size</td>
<td>32B/32B</td>
</tr>
<tr>
<td>iL1/dL1 round-trip latency</td>
<td>2/3 cycles (uncontended)</td>
</tr>
<tr>
<td>iL1/dL1 ports</td>
<td>1 / 2</td>
</tr>
<tr>
<td>iL1/dL1 MSHR entries</td>
<td>16/16</td>
</tr>
<tr>
<td>iL1/dL1 associativity</td>
<td>direct-mapped/4-way</td>
</tr>
<tr>
<td>Memory Disambiguation</td>
<td>Perfect</td>
</tr>
<tr>
<td>Coherence protocol</td>
<td>MESI</td>
</tr>
<tr>
<td>Consistency model</td>
<td>Release consistency</td>
</tr>
</tbody>
</table>
## Experimental Setup

<table>
<thead>
<tr>
<th>Shared L2 Cache Subsystem</th>
<th>4MB, 64B block, 8-way 64</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 MSHR entries</td>
<td>32 cycles (uncontended)</td>
</tr>
<tr>
<td>L2 round-trip latency</td>
<td>64 entries</td>
</tr>
<tr>
<td>Write buffer</td>
<td></td>
</tr>
<tr>
<td><strong>DDR2-800 SDRAM Subsystem [26]</strong></td>
<td></td>
</tr>
<tr>
<td>Transaction Queue</td>
<td>64 entries</td>
</tr>
<tr>
<td>Peak Data Rate</td>
<td>6.4GB/s</td>
</tr>
<tr>
<td>DRAM bus frequency</td>
<td>400 MHz</td>
</tr>
<tr>
<td>Number of Channels</td>
<td>1, 2, 4</td>
</tr>
<tr>
<td>DIMM Configuration</td>
<td>Single rank</td>
</tr>
<tr>
<td>Number of Chips</td>
<td>4 DRAM chips per rank</td>
</tr>
<tr>
<td>Number of Banks</td>
<td>4 per DRAM chip</td>
</tr>
<tr>
<td>Row Buffer Size</td>
<td>2KB</td>
</tr>
<tr>
<td>Address Mapping</td>
<td>Page Interleaving</td>
</tr>
<tr>
<td>Row Policy</td>
<td>Open Page</td>
</tr>
<tr>
<td>tRCD</td>
<td>5 DRAM cycles</td>
</tr>
<tr>
<td>tCL</td>
<td>5 DRAM cycles</td>
</tr>
<tr>
<td>tWL</td>
<td>4 DRAM cycles</td>
</tr>
<tr>
<td>tCCD</td>
<td>4 DRAM cycles</td>
</tr>
<tr>
<td>tWTR</td>
<td>3 DRAM cycles</td>
</tr>
<tr>
<td>tWR</td>
<td>6 DRAM cycles</td>
</tr>
<tr>
<td>tRTP</td>
<td>3 DRAM cycles</td>
</tr>
<tr>
<td>trP</td>
<td>5 DRAM cycles</td>
</tr>
<tr>
<td>tRRD</td>
<td>3 DRAM cycles</td>
</tr>
<tr>
<td>tRAS</td>
<td>18 DRAM cycles</td>
</tr>
<tr>
<td>trC</td>
<td>22 DRAM cycles</td>
</tr>
<tr>
<td>Burst Length</td>
<td>8</td>
</tr>
</tbody>
</table>
## Experimental Setup

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>Problem size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Mining</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCALPARC</td>
<td>Decision Tree</td>
<td>125k pts., 32 attributes</td>
</tr>
<tr>
<td><strong>NAS OpenMP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MG</td>
<td>Multigrid Solver</td>
<td>Class A</td>
</tr>
<tr>
<td>CG</td>
<td>Conjugate Gradient</td>
<td>Class A</td>
</tr>
<tr>
<td><strong>SPEC OpenMP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWIM-OMP</td>
<td>Shallow water model</td>
<td>MinneSpec-Large</td>
</tr>
<tr>
<td>EQUAKE-OMP</td>
<td>Earthquake model</td>
<td>MinneSpec-Large</td>
</tr>
<tr>
<td>ART-OMP</td>
<td>Self-Organizing Map</td>
<td>MinneSpec-Large</td>
</tr>
<tr>
<td><strong>Splash-2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCEAN</td>
<td>Ocean movements</td>
<td>514×514 ocean</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier transform</td>
<td>1M points</td>
</tr>
<tr>
<td>RADIX</td>
<td>Integer radix sort</td>
<td>2M integers</td>
</tr>
</tbody>
</table>
Transaction Queue and L2 Miss Penalties

![Graph showing transaction queue occupancy and L2 miss penalties for various applications.](image)

**Average Transaction Queue Occupancy**

- ART: 40
- CG: 10
- EQUAKE: 20
- FFT: 30
- MG: 20
- OCEAN: 40
- RADIX: 10
- SCALPARC: 20
- SWIM: 28
- MEAN: 10

**Average L2 Load Miss Penalty**

- ART: 1500 (FR-FCFS), 1200 (RL)
- CG: 500 (FR-FCFS), 400 (RL)
- EQUAKE: 600 (FR-FCFS), 500 (RL)
- FFT: 800 (FR-FCFS), 700 (RL)
- MG: 900 (FR-FCFS), 800 (RL)
- OCEAN: 1000 (FR-FCFS), 900 (RL)
- RADIX: 1200 (FR-FCFS), 1100 (RL)
- SCALPARC: 1400 (FR-FCFS), 1300 (RL)
- SWIM: 1500 (FR-FCFS), 1400 (RL)
- MEAN: 1600 (FR-FCFS), 1500 (RL)
More than two Cores?

![Graph showing performance normalized to 4-core system across different numbers of cores for various tasks.

Legend:
- FFT
- CG
- RADIX
- OCEAN
- SWIM
- MG
- SCALPARC
- ART
- EQUAKE

Performance Normalized to 4-Core
FR-FCFS

Number of Cores
4 8 12 16

Presented by Marco Zeller 2018-10-24
Speedup compared to Fair Queuing
Finding the right Parameters

![Graph showing speedup over FR-FCFS for different values of γ and ε.]

Speedup over FR-FCFS

- γ = 0
- γ = 0.5
- γ = 0.95
- γ = 0.99
- γ = 1

Speedup over FR-FCFS

- ε = 0
- ε = 0.05
- ε = 0.1
- ε = 0.2
- ε = 1