Serving DNNs like Clockwork
Performance Predictability from the Bottom Up

Reza Karimi
Ymir Vigfusson
Arpan Gujarati
Safya Alzayat
Wei Hao
Antoine Kaufman
Jonathan Mace
Wei Hao

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

Arpan Gujarati
Safya Alzayat
Wei Hao
Antoine Kaufman
Jonathan Mace

Reza Karimi
Ymir Vigfusson

EMORY UNIVERSITY
Serving DNNs like **Clockwork**
Performance Predictability from the Bottom Up
Serving DNNs like Clockwork
Performance Predictability from the Bottom Up

DNN inference has a very predictable execution time!
Serving DNNs like Clockwork
Performance Predictability from the Bottom Up

DNN inference has a very predictable execution time!

Clockwork
End-to-end predictable DNN serving platform for the Cloud
Serving DNNs like Clockwork
Performance Predictability from the Bottom Up

DNN inference has a very predictable execution time!

Clockwork
End-to-end predictable DNN serving platform for the Cloud

- Supports 1000s of models concurrently per GPU
- Mitigates tail latency, supporting tight latency SLOs (10—100 ms)
- Close to ideal goodput under overload, contention, and bursts
Background
Inference Serving at the Cloud Scale is Difficult
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Periodic
Requests arrive at different rates and regularity

1000s of trained models of different types and resource requirements
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Sustained + High Rate
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Arbitrary
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Each request has an inherent deadline

Latency SLOs (e.g., 100ms)
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Each request has an inherent deadline

Latency SLOs (e.g., 100ms)

<table>
<thead>
<tr>
<th>Model</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>175 ms</td>
<td>6 req/s</td>
</tr>
<tr>
<td>GPU</td>
<td>2.8 ms</td>
<td>350 req/s</td>
</tr>
</tbody>
</table>

HW accelerators are necessary!

Requests arrive at different rates and regularity

Each request has an inherent deadline

Latency SLOs (e.g., 100ms)

<table>
<thead>
<tr>
<th>Model</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>175 ms</td>
<td>6 req/s</td>
</tr>
<tr>
<td>GPU</td>
<td>2.8 ms</td>
<td>350 req/s</td>
</tr>
</tbody>
</table>
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Each request has an inherent deadline

Latency SLOs (e.g., 100ms)

<table>
<thead>
<tr>
<th></th>
<th>Latency</th>
<th>Throughput</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>175 ms</td>
<td>6 req/s</td>
<td>$</td>
</tr>
<tr>
<td>GPU</td>
<td>2.8 ms</td>
<td>350 req/s</td>
<td>$$$</td>
</tr>
</tbody>
</table>

HW accelerators are necessary!
Inference Serving at the Cloud Scale is Difficult

1000s of trained models of different types and resource requirements

Requests arrive at different rates and regularity

Each request has an inherent deadline

Latency SLOs (e.g., 100ms)

Problem
How can cloud providers efficiently share resources while meeting SLOs?

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Latency</th>
<th>Throughput</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>175 ms</td>
<td>6 req/s</td>
<td>$</td>
</tr>
<tr>
<td>GPU</td>
<td>2.8 ms</td>
<td>350 req/s</td>
<td>$$$</td>
</tr>
</tbody>
</table>
Existing Systems Incur Very High Tail Latency
Existing Systems Incur Very High Tail Latency

Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

CDF

Percentile
0 100 500
Latency (ms)

100ms SLO
Existing Systems Incur Very High Tail Latency

Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

![Graph showing tail latency]

Clipper 100ms SLO
Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

Existing Systems Incur Very High Tail Latency

Concurrent DNN inference over GPU
Existing Systems Incur Very High Tail Latency

- Inference latency:
  - 15 trained ResNet50
  - Single GPU worker
  - 16 concurrent requests per model

Tail latency >> SLO

Concurrent DNN inference over GPU

High variance in latency

Throughput gains only 25%
Existing Systems Incur Very High Tail Latency

Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

Concurrent DNN inference over GPU
- High variance in latency
- Throughput gains only 25%

Single-thread latency is extremely predictable
Existing Systems Incur Very High Tail Latency

Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

Concurrent DNN inference over GPU

Preserves DNN predictability at every stage of model serving

Clockwork adopts a contrasting approach!

Single-thread latency is extremely predictable

Concurrent DNN inference gains only 25%

High variance in latency
Existing Systems Incur Very High Tail Latency

Inference latency
- 15 trained ResNet50
- Single GPU worker
- 16 concurrent requests per model

Tail latency >> SLO

Existing Systems Incur Very High Tail Latency

Clockwork adopts a contrasting approach!

Preserves DNN predictability at every stage of model serving

Single-thread latency is extremely predictable

Concurrent DNN inference over GPU
High variance in latency
Throughput gains only 25%

Tail latency within SLO
How does Clockwork Achieve End-to-End Predictability?
Design Principles

Goal: 1000s of models, many users, limited resources
Goal: 1000s of models, many users, limited resources

Maximize sharing
Design Principles

Goal: 1000s of models, many users, limited resources

1. Predictable worker with no choices

Maximize sharing
Design Principles

Goal: 1000s of models, many users, limited resources

1. Predictable worker with no choices

2. Consolidating choices at a central controller

Maximize sharing
Design Principles

Goal: 1000s of models, many users, limited resources

Maximize sharing

1. Predictable worker with no choices

2. Consolidating choices at a central controller

3. Deadline-aware scheduling for SLO compliance
Designing a Predictable Worker (1/2)
Users upload pre-trained models in advance: ⬤ △ ■ ● ★ ◆ ...

Designing a Predictable Worker (1/2)
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ ■ ▢ ▶ □ ■△△△...  

Inference request for ★  

Allocate memory for ★...  

Cold

Worker Node

RAM

GPU Memory

GPU Exec

32 GB

4 TB
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ ■ □ ★ ◆ ...

Inference request for ★ ...

Allocate memory for ★ ...

Execute inference

Allocate memory for ★ ...

GPU Memory

Allocate memory for ★ ...

Execute inference

RAM

GPU

Worker Node

4 TB

32 GB

Users upload pre-trained models in advance: ● △ ■ □ ★ ◆ ...

Inference request for ★ ...

Allocate memory for ★ ...

Execute inference

Allocate memory for ★ ...

Execute inference

RAM

GPU

Worker Node

4 TB

32 GB
Users upload pre-trained models in advance: ○ △ ■ ★ ◆ ...
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ ■ □ ⭐ ...  

Cold

Inference request for ⭐  
Allocate memory for ⭐ ...

Warm

Inference request for ⭐ (execute, since already in GPU memory)  
Execute inference

- RAM
- GPU Memory
- GPU Exec

Worker Node

32 GB

4 TB
Users upload pre-trained models in advance: ● △ ■ ★ ◆ ◆ .....

Inference request for ★ (execute, since already in GPU memory)

Inference request for ★

Allocate memory for ★ ...

Execute inference

Designing a Predictable Worker (1/2)
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ● △ ■ ● ★ ⚫ ...

Queues

Inference request for ★
Allocate memory for ★ ...
Execute inference

Cold

Inference request for ★
(execute, since already in GPU memory)

Warm

RAM

GPU Memory

GPU Exec

Worker Node

4 TB

32 GB

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Users upload pre-trained models in advance: ● △ ■ ● ★ ⚫ ...

Inference request for ★
Allocate memory for ★ ...
Execute inference

Cold

Inference request for ★
(execute, since already in GPU memory)

Warm

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Users upload pre-trained models in advance: ● △ ■ ● ★ ⚫ ...

Inference request for ★
Allocate memory for ★ ...
Execute inference

Cold

Inference request for ★
(execute, since already in GPU memory)

Warm

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Users upload pre-trained models in advance: ● △ ■ ● ★ ⚫ ...

Inference request for ★
Allocate memory for ★ ...
Execute inference

Cold

Inference request for ★
(execute, since already in GPU memory)

Warm

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Users upload pre-trained models in advance: ● △ ■ ● ★ ⚫ ...

Inference request for ★
Allocate memory for ★ ...
Execute inference

Cold

Inference request for ★
(execute, since already in GPU memory)

Warm

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times
Designing a Predictable Worker (1/2)

Users upload pre-trained models in advance: ○ △ ■ ★ ⭐ ...  

Inference request for ★ (execute, since already in GPU memory)

Inference request for ★...

Allocate memory for ★ ...

Execute inference

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Managed memory can be unpredictable
- GPU memory (cache) hits & misses

ResNet-50 — Hit: 2.3 ms | Miss: 10.6 ms

Queues

RAM

GPU Memory

GPU Exec

Worker Node

32 GB

4 TB
Designing a Predictable Worker (2/2)
Designing a Predictable Worker (2/2)

Predictable Clockwork worker process
Designing a Predictable Worker (2/2)

Predictable Clockwork worker process

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Solution

Execute inference one at a time
Designing a Predictable Worker (2/2)

Managed memory can be unpredictable

Solution
Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Predictable Clockwork worker process

Concurrent inferences
Proprietary & undocumented policies
Unpredictable response times

Solution
Execute inference one at a time
Designing a Predictable Worker (2/2)

Managed memory can be unpredictable

Solution
Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Predictable Clockwork worker process

Concurrent inferences

Proprietary & undocumented policies
Unpredictable response times

Solution
Execute inference one at a time
Designing a Predictable Worker (2/2)

Managed memory can be unpredictable

Solution
Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Choices outsourced via action APIs

Predictable Clockwork worker process

LOAD/UNLOAD (◇, Deadline)

INFER (★, I/P, Deadline)

Managed memory can be unpredictable

Solution
Preallocate GPU memory & manage it explicitly using LOAD/UNLOAD actions

Concurrent inferences

Proprietary & undocumented policies

Unpredictable response times

Solution
Execute inference one at a time
Consolidating Choices
Consolidating Choices

Users → Centralized Controller → Worker processes

Worker processes:
- LOADs
- INFERs

Centralized Controller:

GPU Worker Node W₁

RAM
GPU Memory
PageCache
GPU Exec
Consolidating Choices

Global State Manager

Latency Profiles
Pending Tasks
Memory State

Centralized Controller
Worker processes

Users

GPU Worker Node W₁

RAM
PageCache
GPU Memory
GPU Exec
Users -> Consolidating Choices

Centralized Controller -> Worker processes

Global State Manager
- Latency Profiles
- Pending Tasks
- Memory State

Smarter load balancing & scheduling decisions

GLOBAL LOADs
GLOBAL INFERs

RAM
GPU Memory
PageCache
GPU Exec

GPU Worker Node W1
SLO-aware Scheduling
SLO-aware Scheduling

Users → Centralized Controller → Worker processes

Centralized Controller

Worker processes

LOADs → INFERs

GPU Memory → GPU Exec

GPU Worker Node $W_1$

Users: Users request tasks.
Centralized Controller: Manages task allocation.
Worker processes: Execute tasks on worker nodes.
LOADs: Load operations.
INFERs: Inference operations.
GPU Memory: GPU memory management.
GPU Exec: GPU execution.
GPU Worker Node $W_1$: Specific GPU worker node.
SLO-aware Scheduling

Users

Centralized Controller

Worker processes

GPU

Worker Node

W1

GPU

Pending Tasks

Time

W1

now

free

GPU Exec

GPU Memory

LOADs

INFERs

RAM
SLO-aware Scheduling

Centralized Controller

Worker processes

GPU Worker Node \( W_1 \)

Pending Tasks

Time

W_1 GPU

Inference request for

Since \( t_{\text{deadline}} < t_{\text{free}} \), inference request for \( \circ \) is cancelled
SLO-aware Scheduling

Centralized Controller

Worker processes

GPU Worker Node W₁

Inference request for ⭐️

Deadline is further away

Pending Tasks

Time

W₁

GPU

CPU

LOADs

INFERs

Deadline is further away

Page Cache

Time

now

free

deadline
SLO-aware Scheduling

Users → Centralized Controller → Worker processes

From latency profiles

Pending Tasks

Deadline is further away

Inference request for ✶

Time

W₁ GPU

Deadline is further away

Loads

Infer

Δ infer

Deadline is further away

From latency profiles

Workers

W₁ GPU

INFERs

GPU Exec

GPU Memory

RAM

GPU

Inference request for ✶
SLO-aware Scheduling

From latency profiles

Deadline is further away

Since $t_{\text{free}} + \Delta_{\text{infer}} < t_{\text{deadline}}$, inference request for $\star$ is scheduled on $W_1$
SLO-aware Scheduling

What if \( \Delta \) does not finish on time?
SLO-aware Scheduling

Users → Centralized Controller → Worker processes

-**W***<sub>1</sub> Worker Node

- **RAM**, **GPU Memory**, **GPU Exec**

**Pending Tasks**

- **Δ***<sub>infer</sub>

- **t***<sub>now</sub>, **t***<sub>free</sub>, **t***<sub>deadline</sub>, **t***<sub>latest</sub>

**What if Δ does not finish on time?**

**Clockwork also tracks t***<sub>latest</sub>, and cancels if it fails to start before t***<sub>latest</sub>
SLO-aware Scheduling

Many benefits
- Prevent wasteful work
- Manage LOAD → INFER dependencies
- Choosing the best batching strategy
Evaluation
Questions
How does Clockwork compare to prior model serving systems Clipper and INFaaS?
How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?
Questions

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?
Questions

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?
Questions

Simple workloads in controlled settings

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?
Questions

Simple workloads in controlled settings

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?

Are Clockwork workers predictable?

Does consolidating choice help achieve end-to-end predictability?

Can Clockwork controller Scale?

Workloads from production traces
Questions

Simple workloads in controlled settings

How does Clockwork compare to prior model serving systems Clipper and INFaaS?

Can Clockwork serve thousands of model instances?

How low can Clockwork go in terms of the latency SLOs it can satisfy?

Can Clockwork isolate the performance of latency-sensitive clients from batch requests without latency SLOs?

This talk

Are Clockwork workers predictable?

Does consolidating choice help achieve end-to-end predictability?

Can Clockwork controller Scale?

Workloads from production traces
Experiment Setup

12 Workers: NVIDIA Tesla v100 GPU | 32 GB GPU Memory + 1 Controller + 1 Client
Experiment Setup

12 Workers: NVIDIA Tesla v100 GPU | 32 GB GPU Memory + 1 Controller + 1 Client

Microsoft’s Azure Functions

Shahrad et al. “Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider.” USENIX ATC 2020

Workload

46,000 functions, 2 weeks
- Heavy sustained workloads
- Low utilization cold workloads
- Workloads with periodic spikes
- Bursty workloads
Experiment Setup

12 Workers: NVIDIA Tesla v100 GPU | 32 GB GPU Memory + 1 Controller + 1 Client

Microsoft’s Azure Functions
Shahrad et al. “Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider.” USENIX ATC 2020

4026 model instances
- Saturates 768 GB RAM
- 61 different model architectures
- ResNet, DenseNet, Inception, etc.

46,000 functions, 2 weeks
- Heavy sustained workloads
- Low utilization cold workloads
- Workloads with periodic spikes
- Bursty workloads

Workload

Rate vs. Time Graph
Are Clockwork Workers Predictable?
Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling.

- **Overpredictions** → Idle resources
- **Underpredictions** → SLO violations
Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling.

- Overpredictions → Idle resources
- Underpredictions → SLO violations

Experiment duration = 6 hours,
Offered load ~ 10,000 r/s
Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling.

Overpredictions → Idle resources
Underpredictions → SLO violations

Clockwork consistently overpredicts more than its underpredicts.
Are Clockwork Workers Predictable?

Clockwork relies on predicting the model inference latency for scheduling

Overpredictions → Idle resources
Underpredictions → SLO violations

Clockwork consistently overpredicts more than its underpredicts
Errors are significant only in extremely rare cases

Underprediction error = 55us
Overprediction error = 144us
Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s
Latency SLO = 100 ms deadline for each request
Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s
Latency SLO = 100 ms deadline for each request

Goodput = SLO compliant throughput

\[
\text{Goodput} = \frac{\text{Throughput}}{\text{SLO}}
\]
Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s
Latency SLO = 100 ms deadline for each request

Goodput = SLO compliant throughput

Latency of all completed requests

Time (Minutes)

0 60 120 180 240 300 360

12000 10000 8000 6000 4000 2000 0

Offered Load

Goodput

Maximum

99th %ile

Median

Latency (ms)
Does Consolidating Choice Help?

- Offered load ~10,000 r/s, periodic spikes ~12,000 r/s
- Latency SLO = 100 ms deadline for each request

The workload is successfully scheduled by Clockwork:
- Goodput ≈ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO
Does Consolidating Choice Help?

Offered load \(\sim 10,000\) r/s, periodic spikes \(\sim 12,000\) r/s

Latency SLO = 100 ms deadline for each request

The workload is successfully scheduled by Clockwork

- Goodput \(\approx\) offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO
Does Consolidating Choice Help?

Latency SLO = 100 ms deadline for each request

The workload is successfully scheduled by Clockwork
- Goodput $\approx$ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO
Does Consolidating Choice Help?

Offered load ~10,000 r/s, periodic spikes ~12,000 r/s
Latency SLO = 100 ms deadline for each request

- Goodput ≈ offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO
Does Consolidating Choice Help?

- Offered load ~10,000 r/s, periodic spikes ~12,000 r/s
- Latency SLO = 100 ms deadline for each request

The workload is successfully scheduled by Clockwork:
- Goodput \(\approx\) offered load
- Out of 208 million requests, only 58 failed due to mispredictions
- All others completed within SLO

Goodput = SLO compliant throughput

Latency of all completed requests

- Batch Size
- Batching prioritized, absorbs spikes
- Many cold starts
- Cold requests = 1.3% of all requests
- Coldstart
Does Clockwork Controller Scale?
Does Clockwork Controller Scale?

Methodology

- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers
Does Clockwork Controller Scale?

**Methodology**
- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers
Does Clockwork Controller Scale?

**Methodology**
- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers
Methodology
- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers
Does Clockwork Controller Scale?

**Methodology**
- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers

**Graph**
- **Linear scalability until #workers = 110**
- Goodput limited by worker’s utilization
Does Clockwork Controller Scale?

**Methodology**
- Replace GPU workers with emulated workers
- From the controller’s vantage point, nothing changes
- Measure the peak goodput as we vary #workers

**Maximum goodput:**
103,387 r/s for 110 workers

**Linear scalability until #workers = 110**
Goodput limited by worker’s utilization

**Bottleneck shifts to Clockwork**
Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability
- Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice
Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability
- Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice

Clockwork: From DNN predictability to an E2E predictable DNN serving platform
- Recursively ensures that all internal architecture components have predictable performance
- Concentrating all choices in a centralized controller
Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability
- Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice

Clockwork: From DNN predictability to an E2E predictable DNN serving platform
- Recursively ensures that all internal architecture components have predictable performance
- Concentrating all choices in a centralized controller

Outperforms state-of-the-art DNN serving platforms
- Efficiently fulfills aggressive tail-latency SLOs
- Supports 1000s of DNN models with varying workload characteristics concurrently on each GPU
Summary

Key idea: DNN executions on GPUs exhibit negligible latency variability
- Intuitive – DNN inferences involve no conditional branches – and demonstrable in practice

Clockwork: From DNN predictability to an E2E predictable DNN serving platform
- Recursively ensures that all internal architecture components have predictable performance
- Concentrating all choices in a centralized controller

Outperforms state-of-the-art DNN serving platforms
- Efficiently fulfills aggressive tail-latency SLOs
- Supports 1000s of DNN models with varying workload characteristics concurrently on each GPU

https://gitlab.mpi-sws.org/cld/ml/clockwork