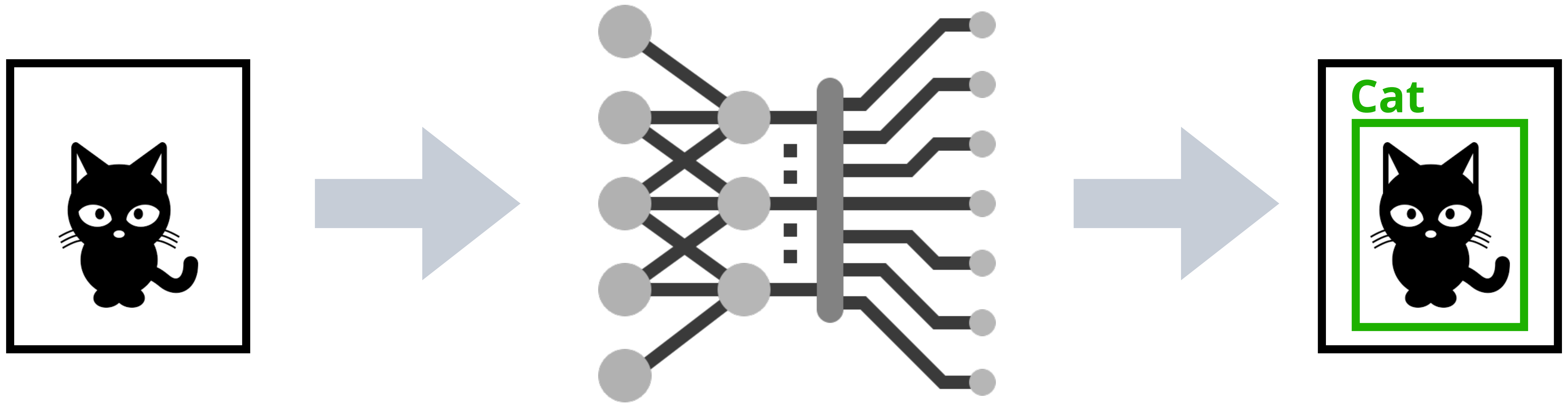
Jellyfish: Timely Inference Serving for Dynamic Edge Networks

Vinod Nigade, Pablo Bauszat, Henri Bal, Lin Wang
Vrije Universiteit Amsterdam

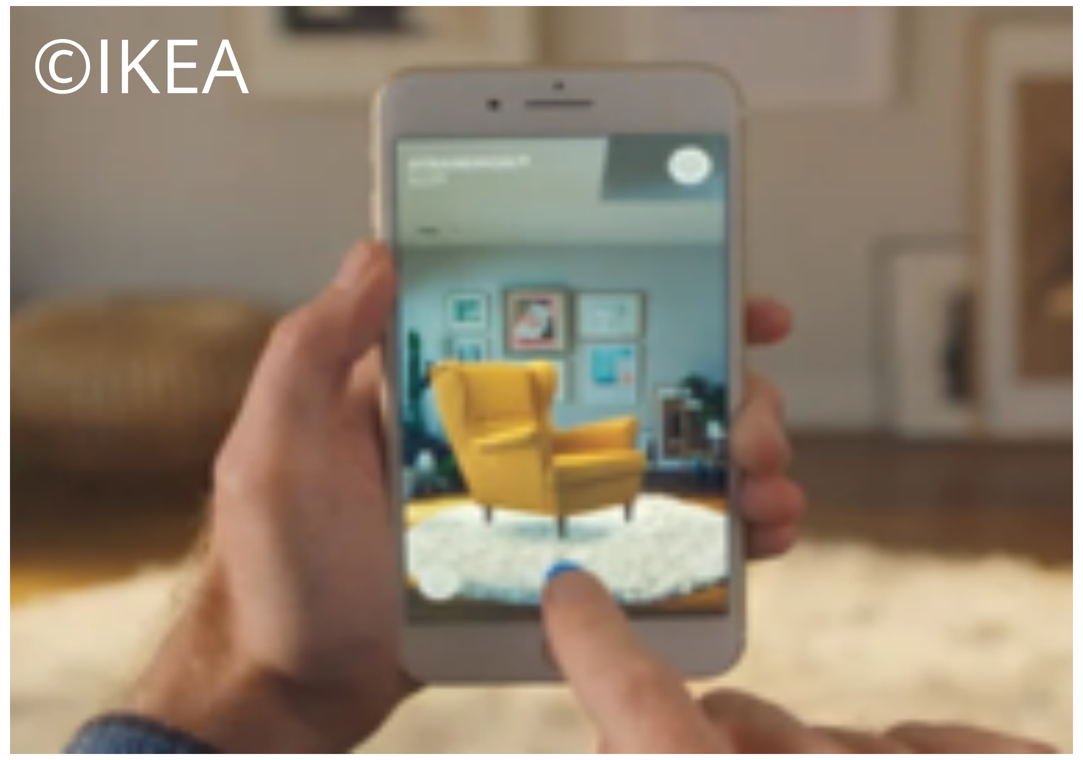
IEEE RTSS 2022

DNNs are becoming a critical part of modern applications

Deep Neural Networks (DNNs)



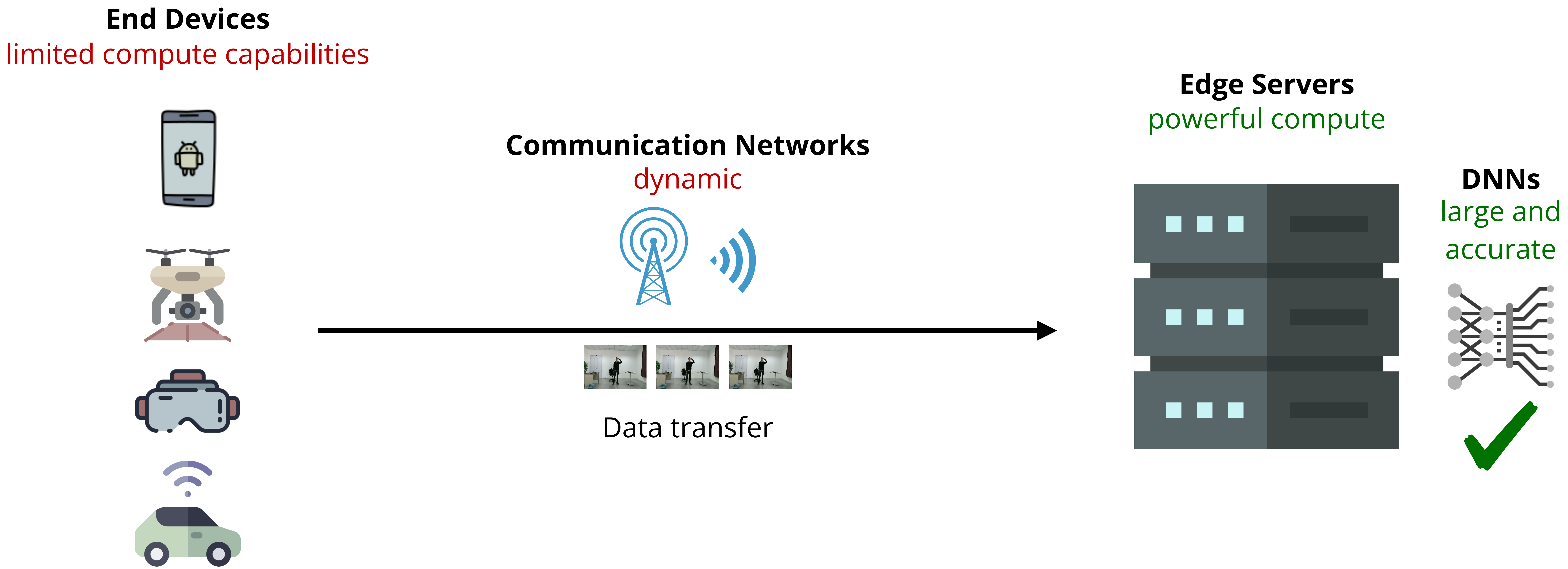
Applications



Augmented Reality

Autonomous Driving

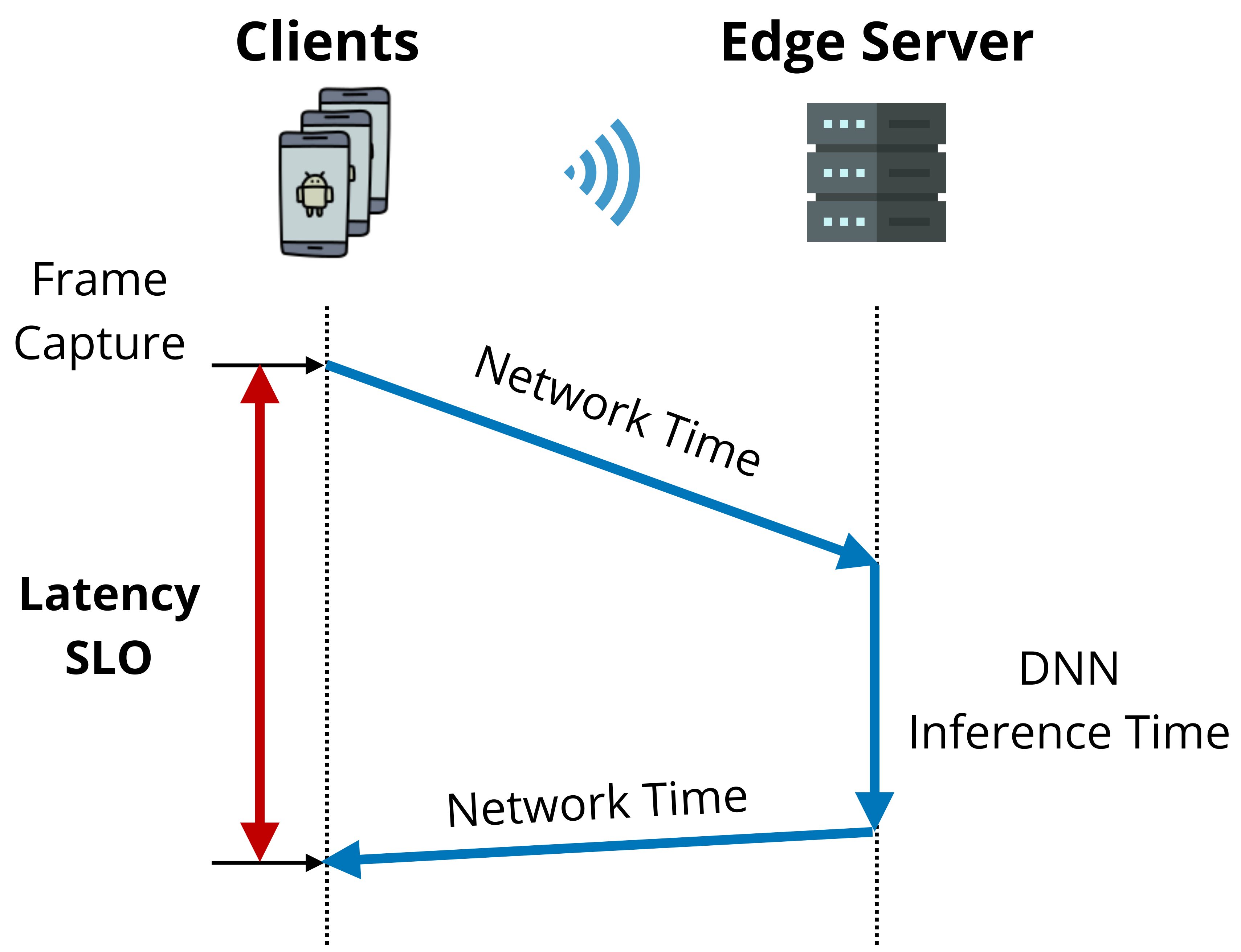
Applications have to offload DNNs to edge servers



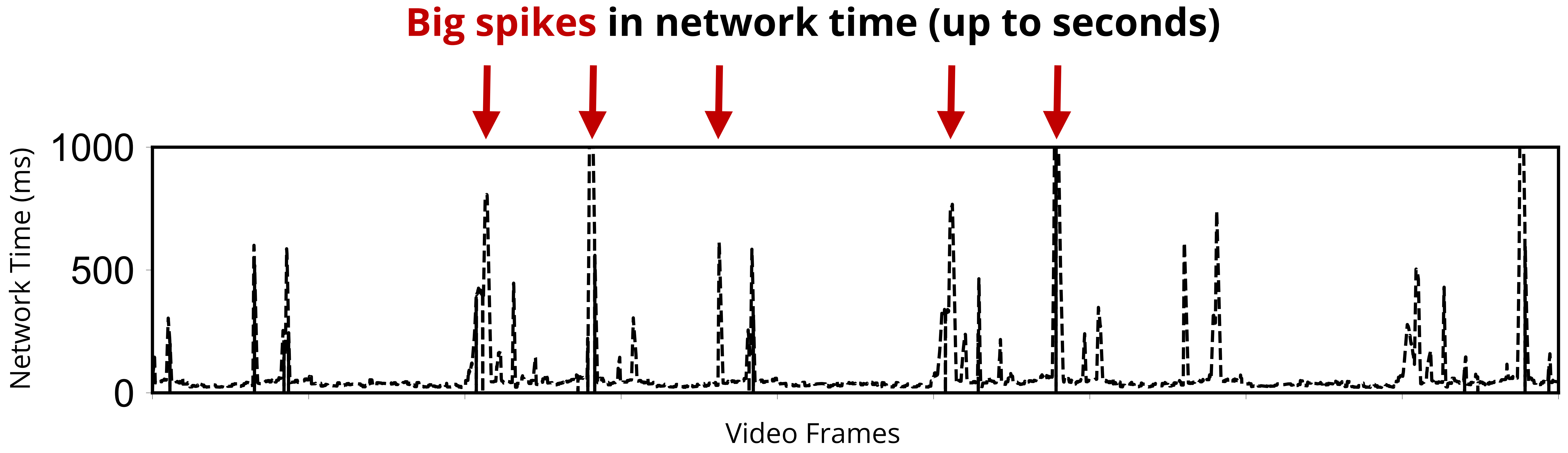
Applications need timely predictions

Edge-serving systems should support…

- end-to-end **latency service-level objectives (SLOs)** (e.g., 100ms) that include network time to transfer data
- the application's desired **request rate** (e.g., 25 FPS)
- **multiple clients** and their **aggregate request rate** on fixed compute resources, e.g., via request batching



Data transfer from clients shows significant variable delays



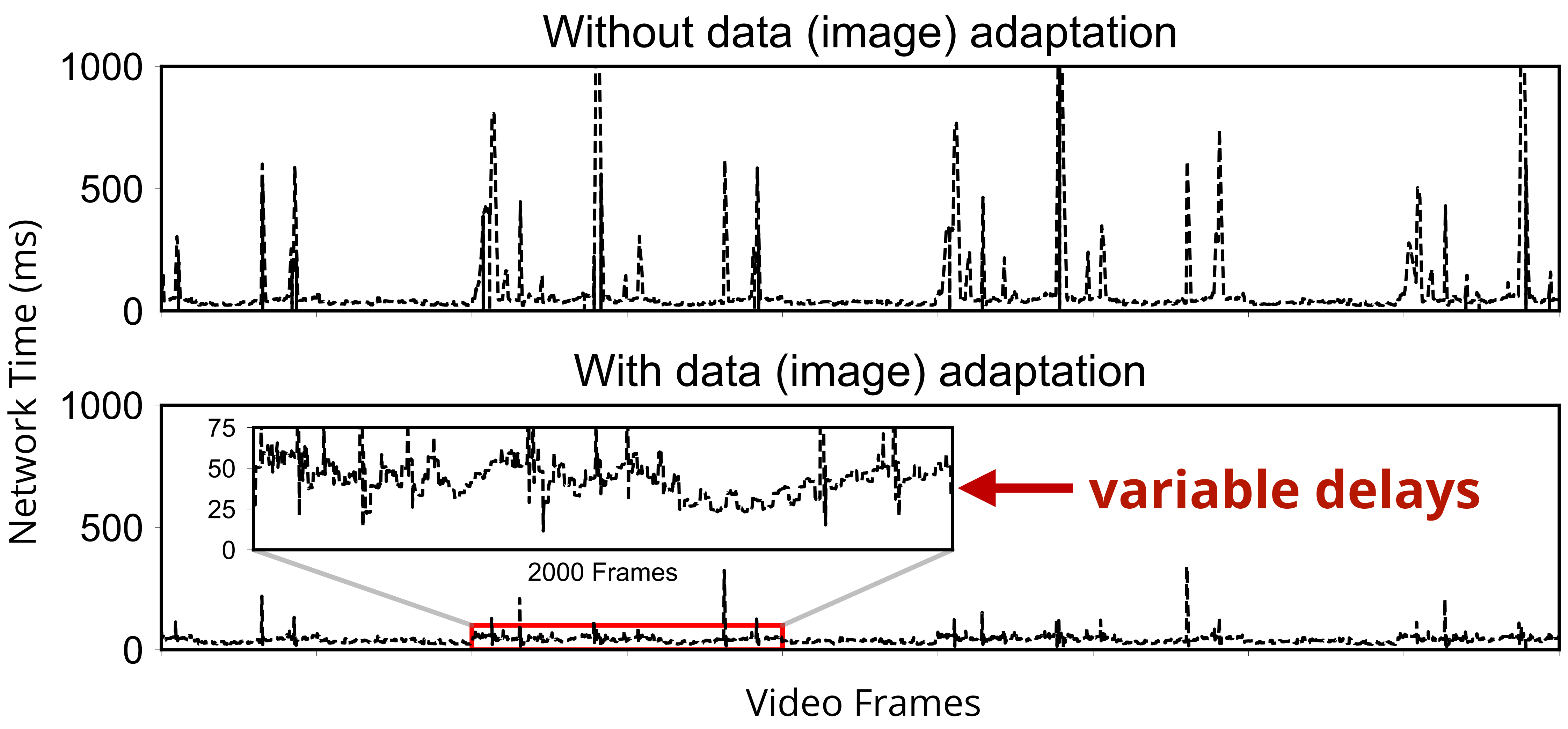
Data transfer over a network connection emulated with an LTE trace

How to handle variable network delays to serve requests on time?

Use data adaptation on the client side

- Adapt the data size based on the available network bandwidth (e.g., [AWStream, SIGCOMM'18])

  \+ Smooths out big spikes leading to more **stable throughput**
  \- Still significant variable delays causing **variable compute budget** on the server side

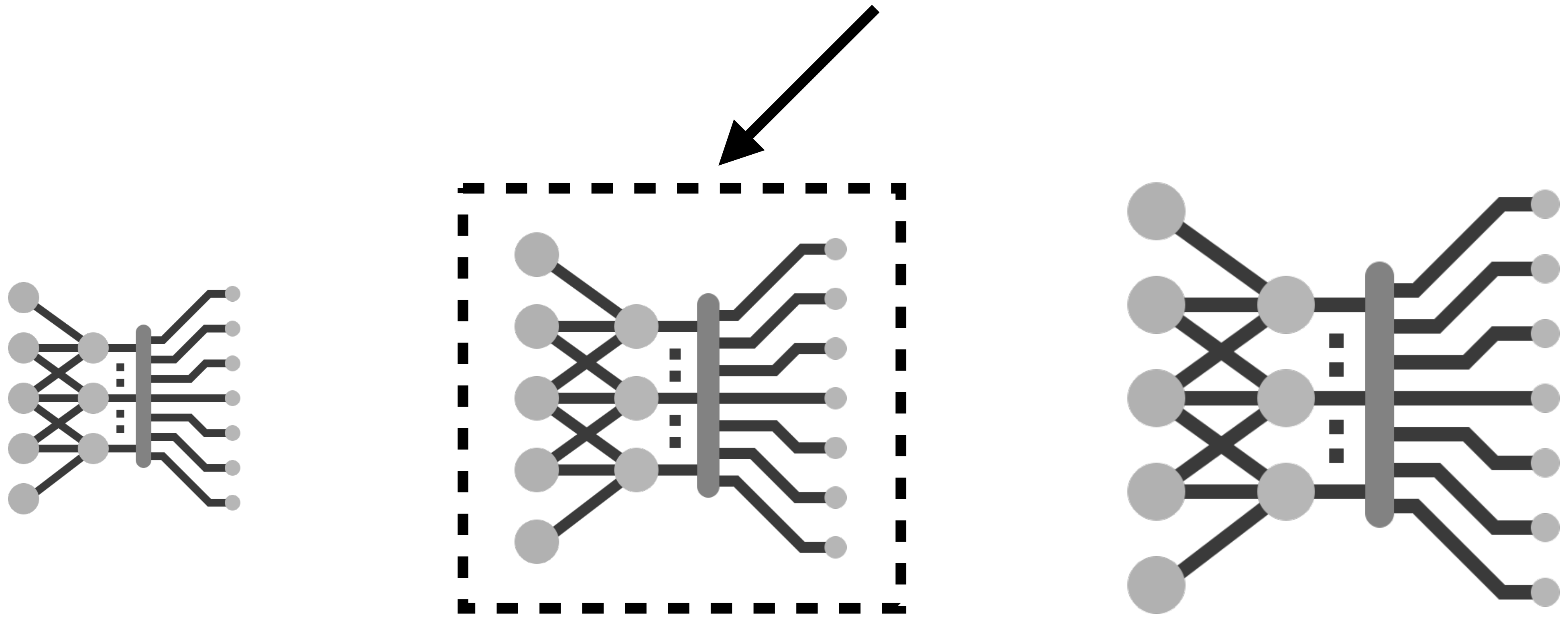


How to timely serve inference requests given a variable compute budget?

Use DNN adaptation on the server side

- Deploy DNN variants with different **latency-accuracy tradeoff profiles**
- Select a DNN variant for a given **compute budget**

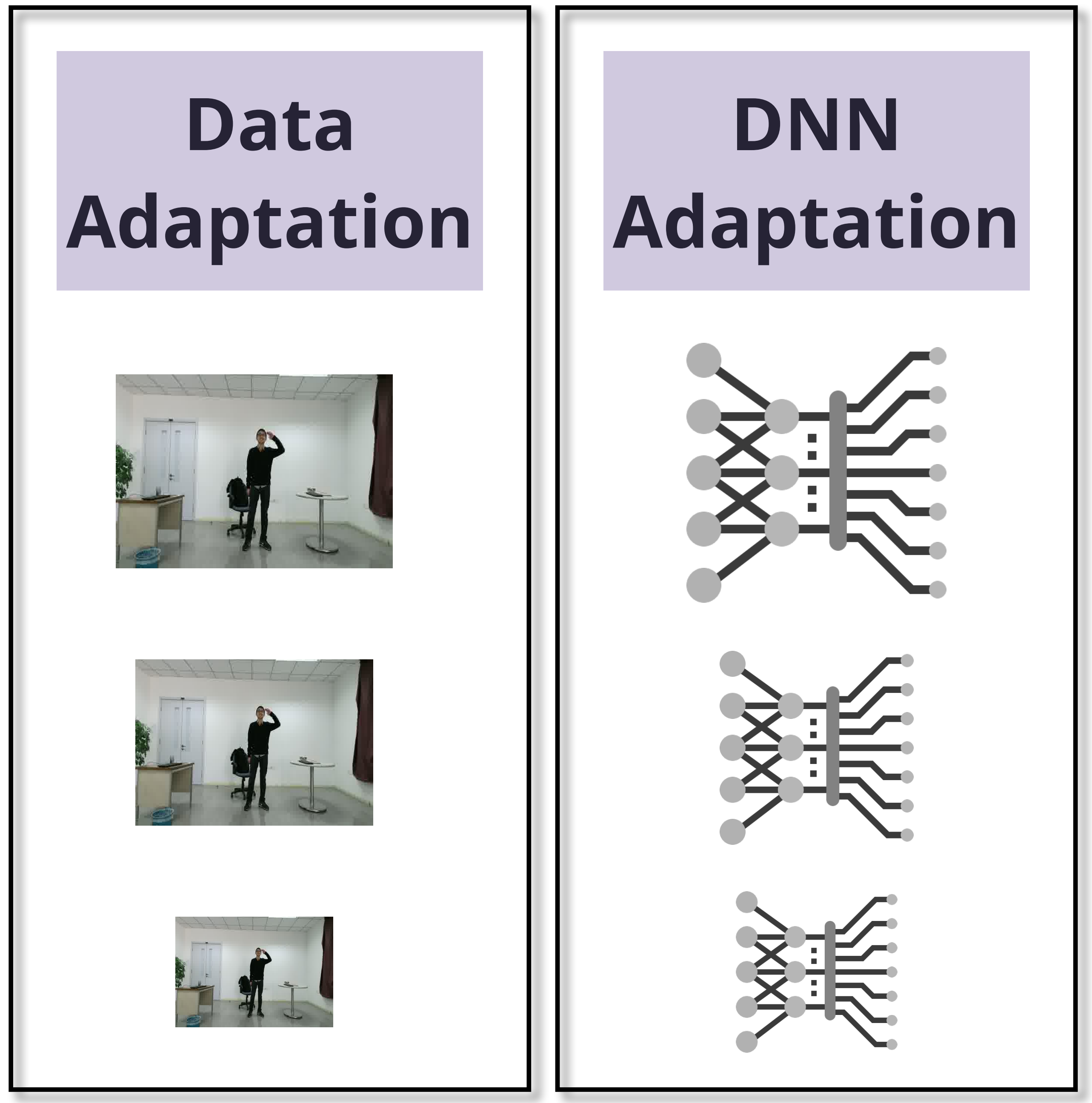
DNN variant selected when the compute budget time is **< 100ms** and **> 50ms**



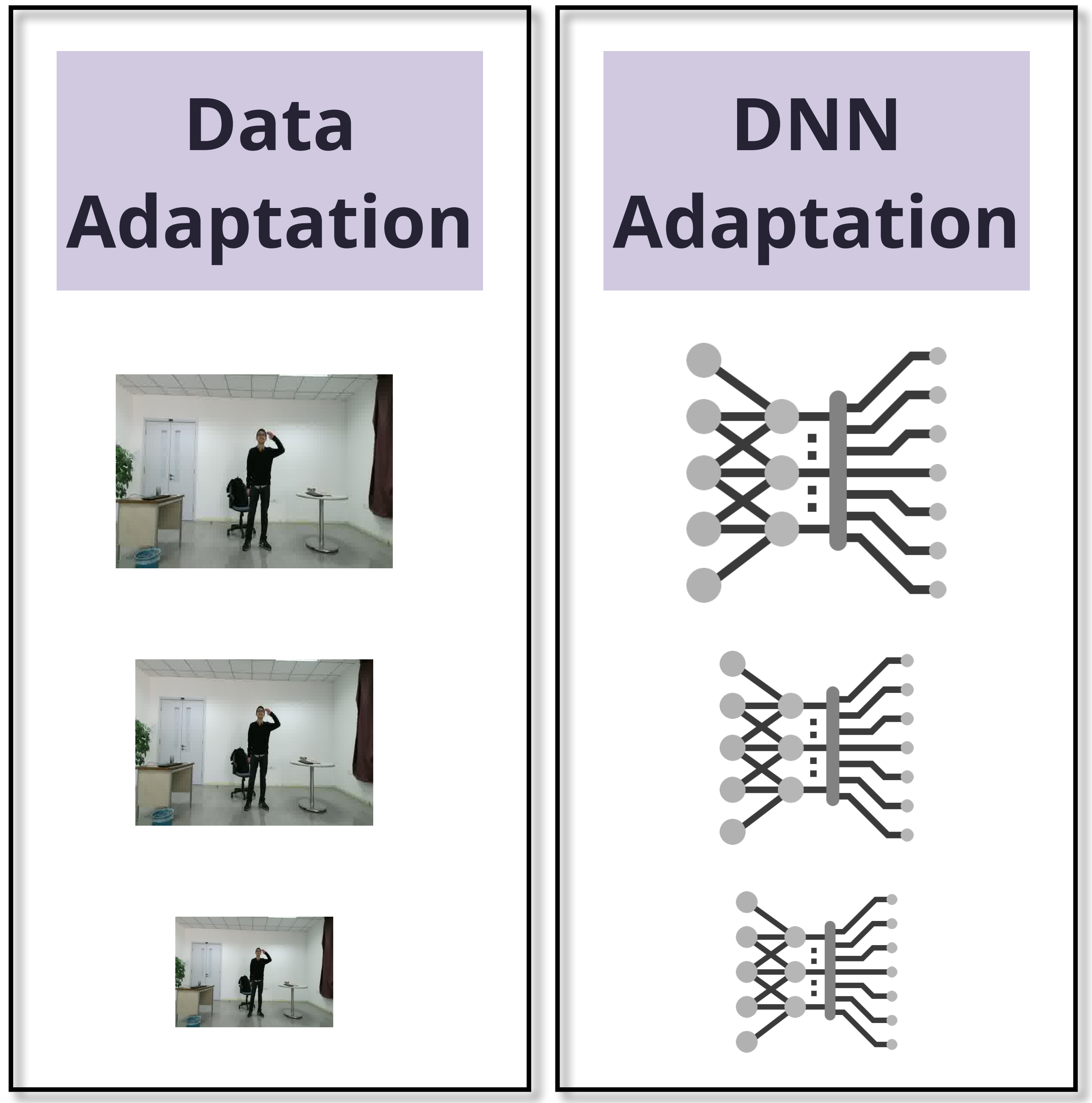
DNN variants			
Inference Latency	10ms	50ms	100ms

e.g., [ALERT, ATC’20] [SubFlow, RTAS’20]

Challenges in combining data and DNN adaptation



Challenges in combining data and DNN adaptation

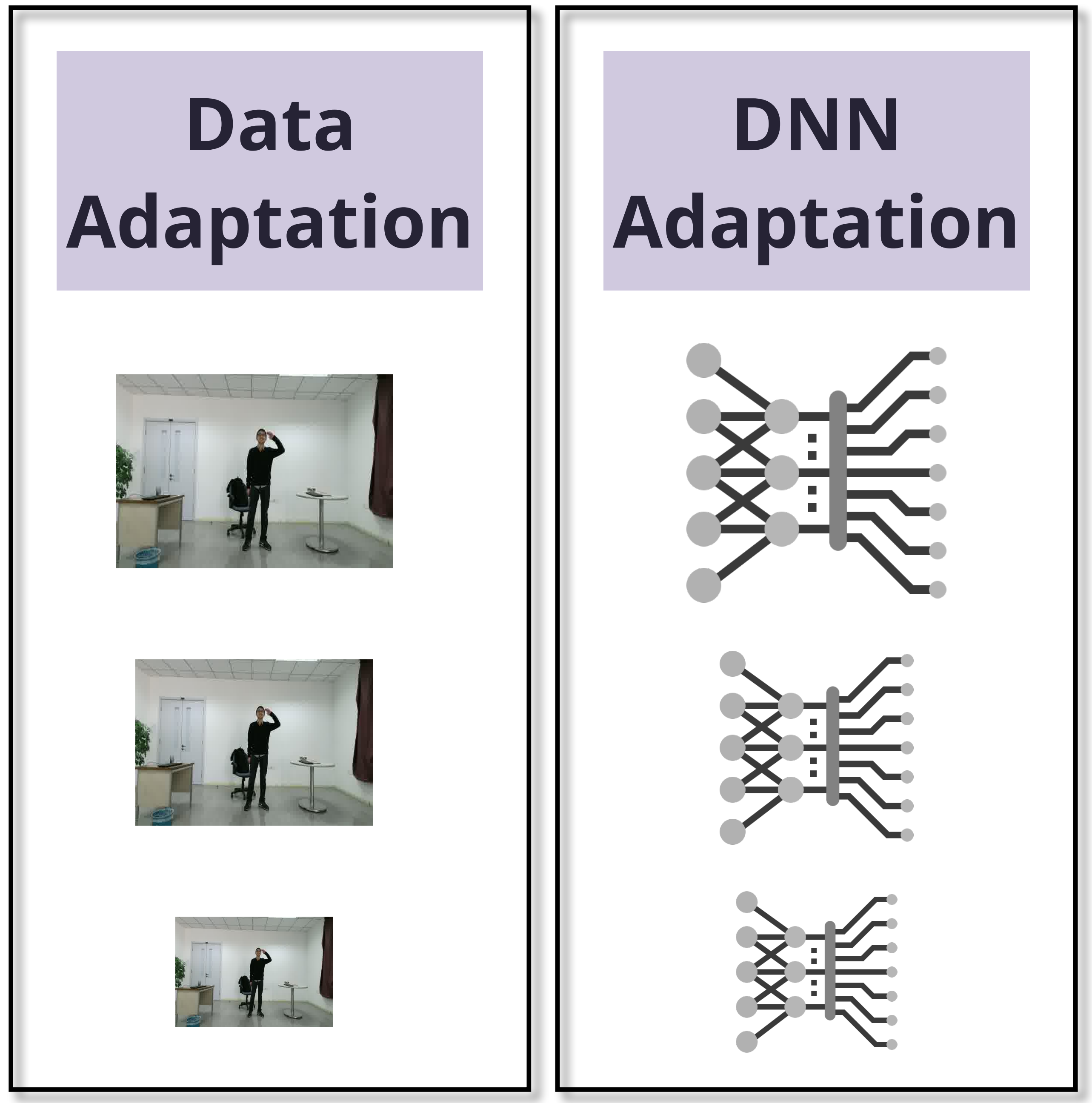


- **C1 Misaligned adaptation decisions**

Case 1: Bigger data size and smaller DNN input size

Leads to a waste of extra network time (100-150ms)

Challenges in combining data and DNN adaptation



- **C1 Misaligned adaptation decisions**

Case 1: Bigger data size and smaller DNN input size

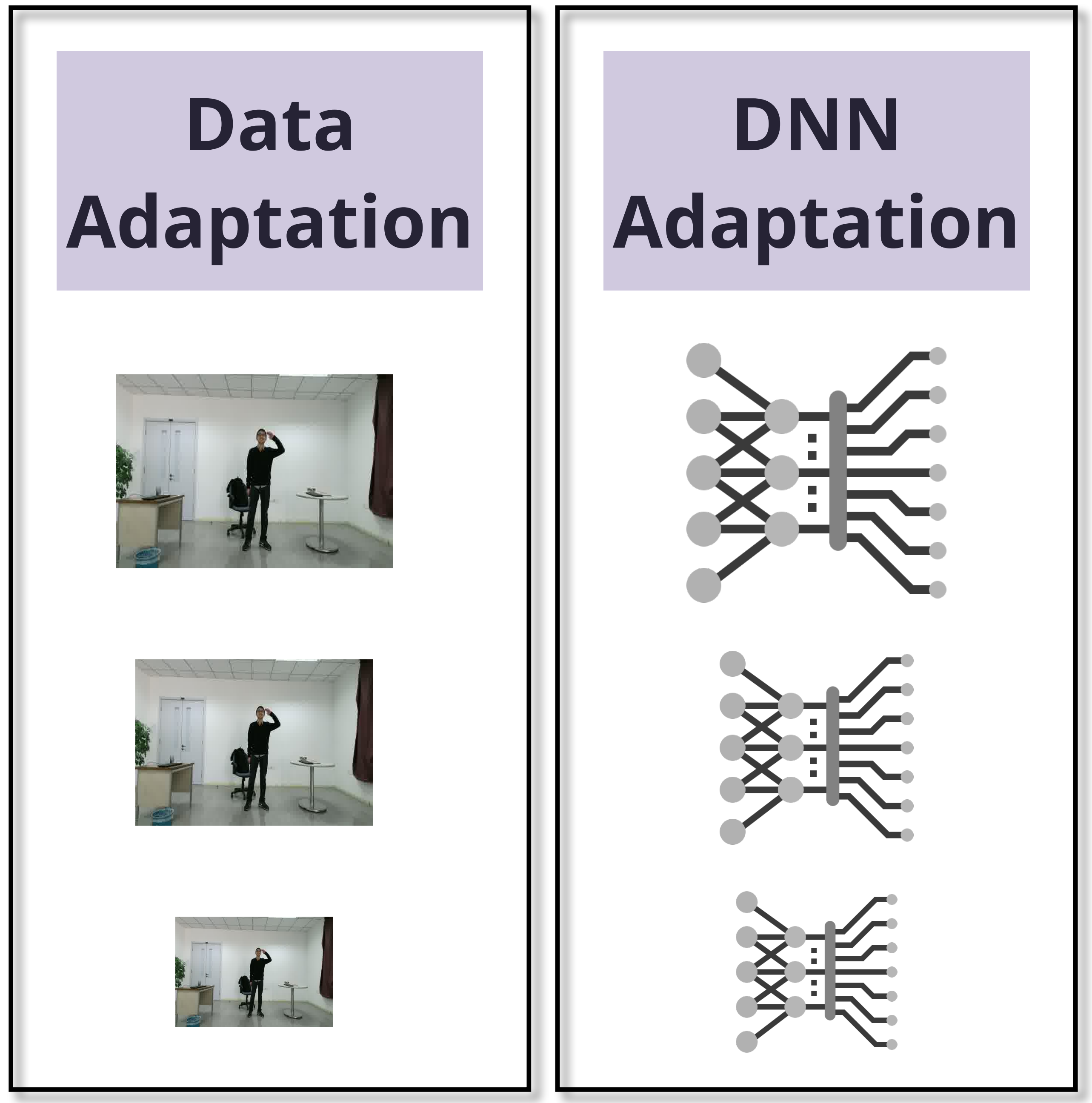
Leads to a waste of extra network time (100-150ms)

Case 2: Smaller data size and bigger DNN input size

Leads to accuracy degradation[1]

[1] [Dengxin Dai, et. al., WACV'16]

Challenges in combining data and DNN adaptation



- **C1 Misaligned adaptation decisions**

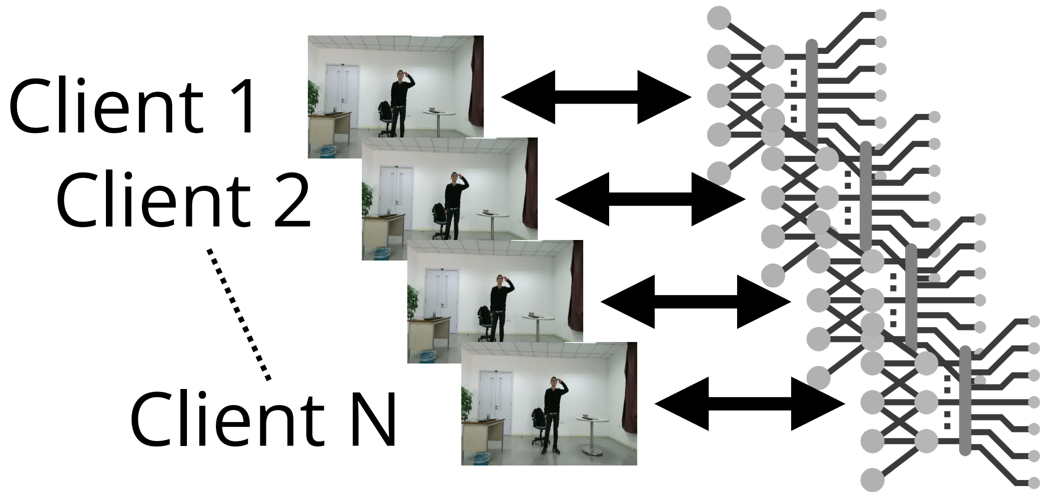
Case 1: Bigger data size and smaller DNN input size

Leads to a waste of extra network time (100-150ms)

Case 2: Smaller data size and bigger DNN input size

Leads to accuracy degradation[1]

- **C2 Un-coordinated adaptations for multiple clients**



No resource capacity to run separate DNNs for every client

[1] [Dengxin Dai, et. al., WACV'16]

Introducing…

Jellyfish

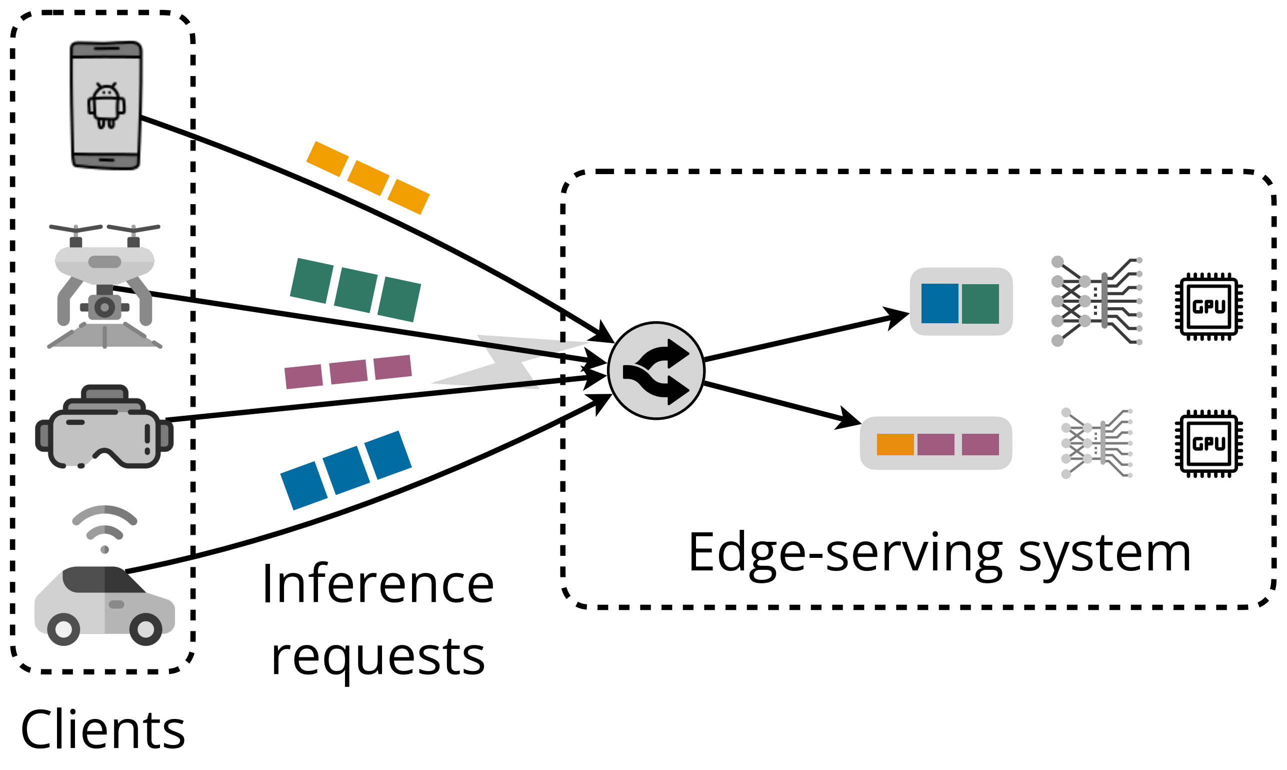
An **edge-centric serving system** for dynamic edge networks with **timeliness** as a goal

- Defines latency SLO in an end-to-end fashion, taking into account the **variable network time**

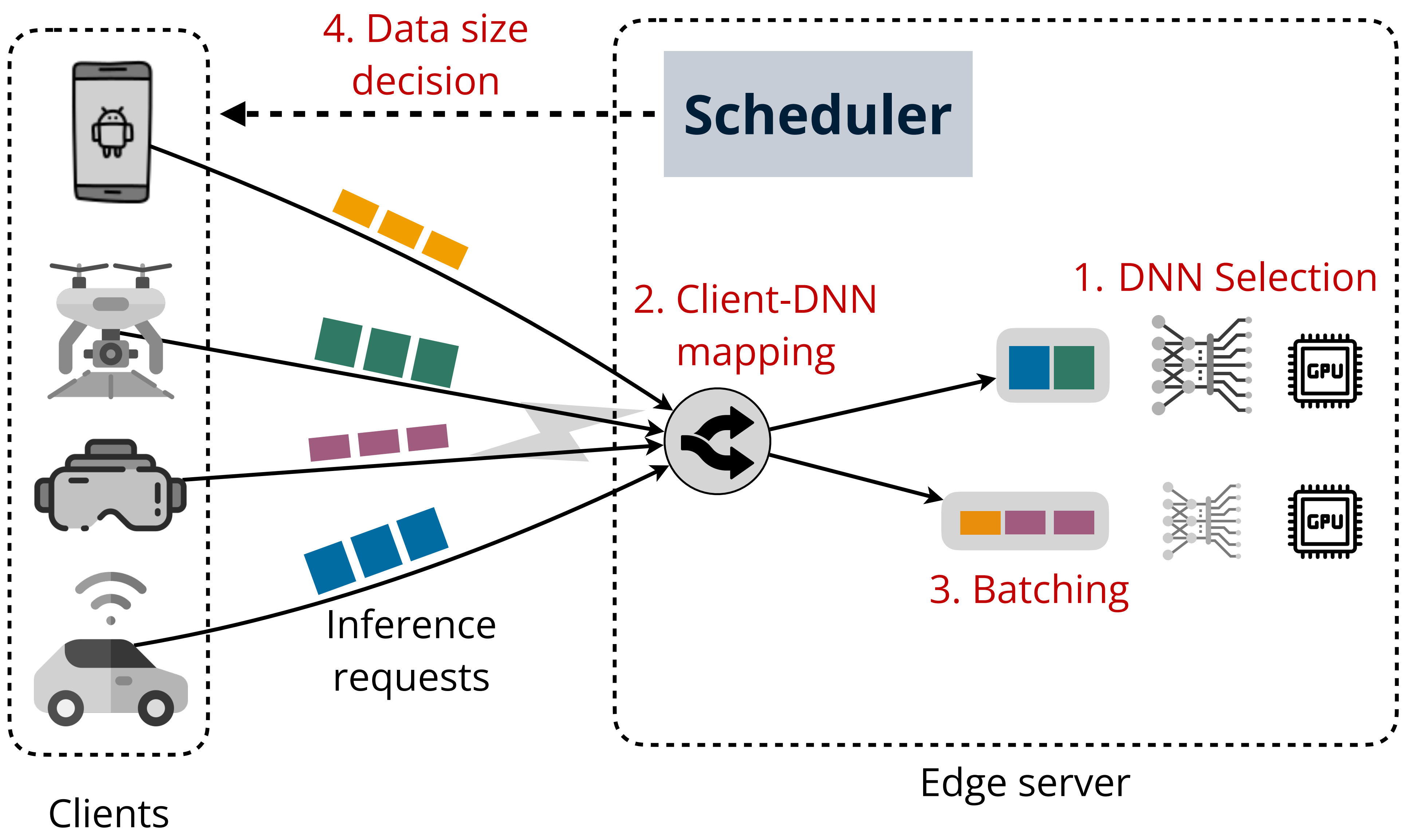
- Utilizes **data and DNN adaptation jointly** and aligns their adaptation decisions

- Coordinates adaptation decisions for multiple clients, a.k.a. **collective adaptation**

- Supports **batching** for resource efficiency



Jellyfish has to solve a complex scheduling problem



The scheduling problem involves **multiple complex steps**

1. Selection of a few DNN variants on a limited amount of compute resources

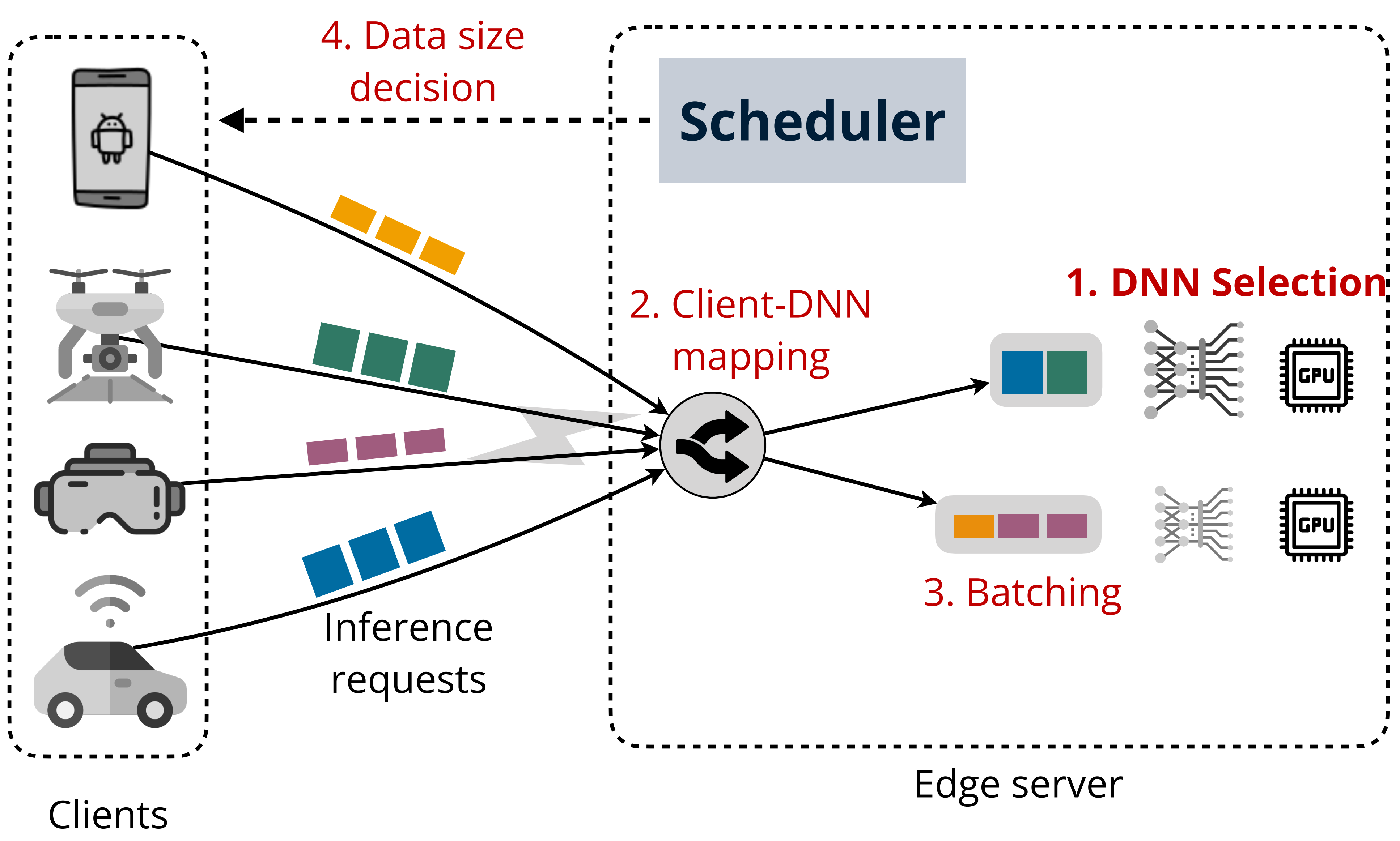
2. Mapping every client (their requests) to the selected DNN variants

3. Deciding the batch size of every DNN variant for serving multiple clients

4. Informing clients about their mapped DNN and data sizes

Solve continuously without violating end-to-end latency SLO

Jellyfish has to solve a complex scheduling problem

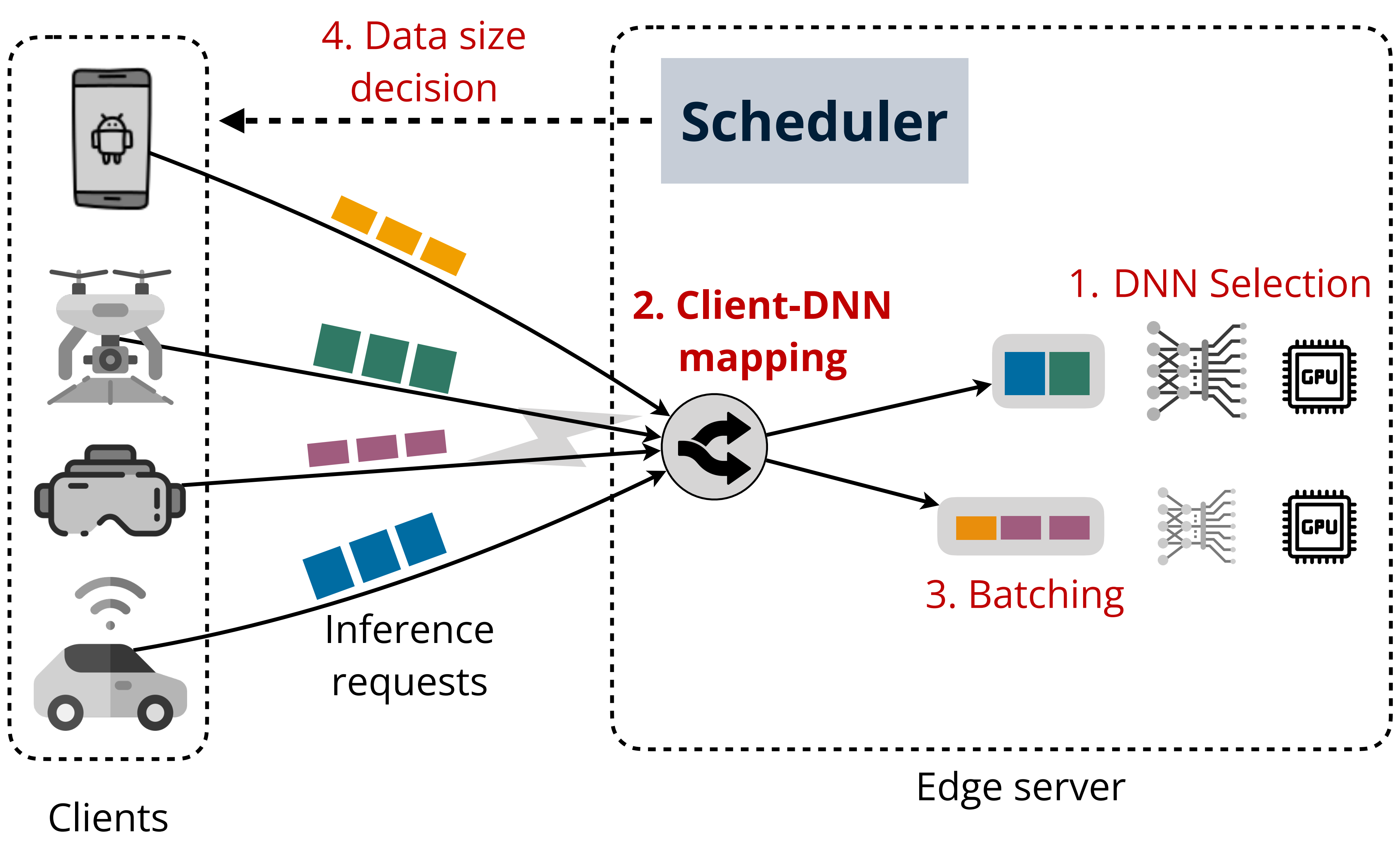


The scheduling problem involves **multiple complex steps**

1. **Selection of a few DNN variants on a limited amount of compute resources**
2. Mapping every client (their requests) to the selected DNN variants
3. Deciding the batch size of every DNN variant for serving multiple clients
4. Informing clients about their mapped DNN and data sizes

Solve continuously without violating end-to-end latency SLO

Jellyfish has to solve a complex scheduling problem



The scheduling problem involves **multiple complex steps**

1. Selection of a few DNN variants on a limited amount of compute resources

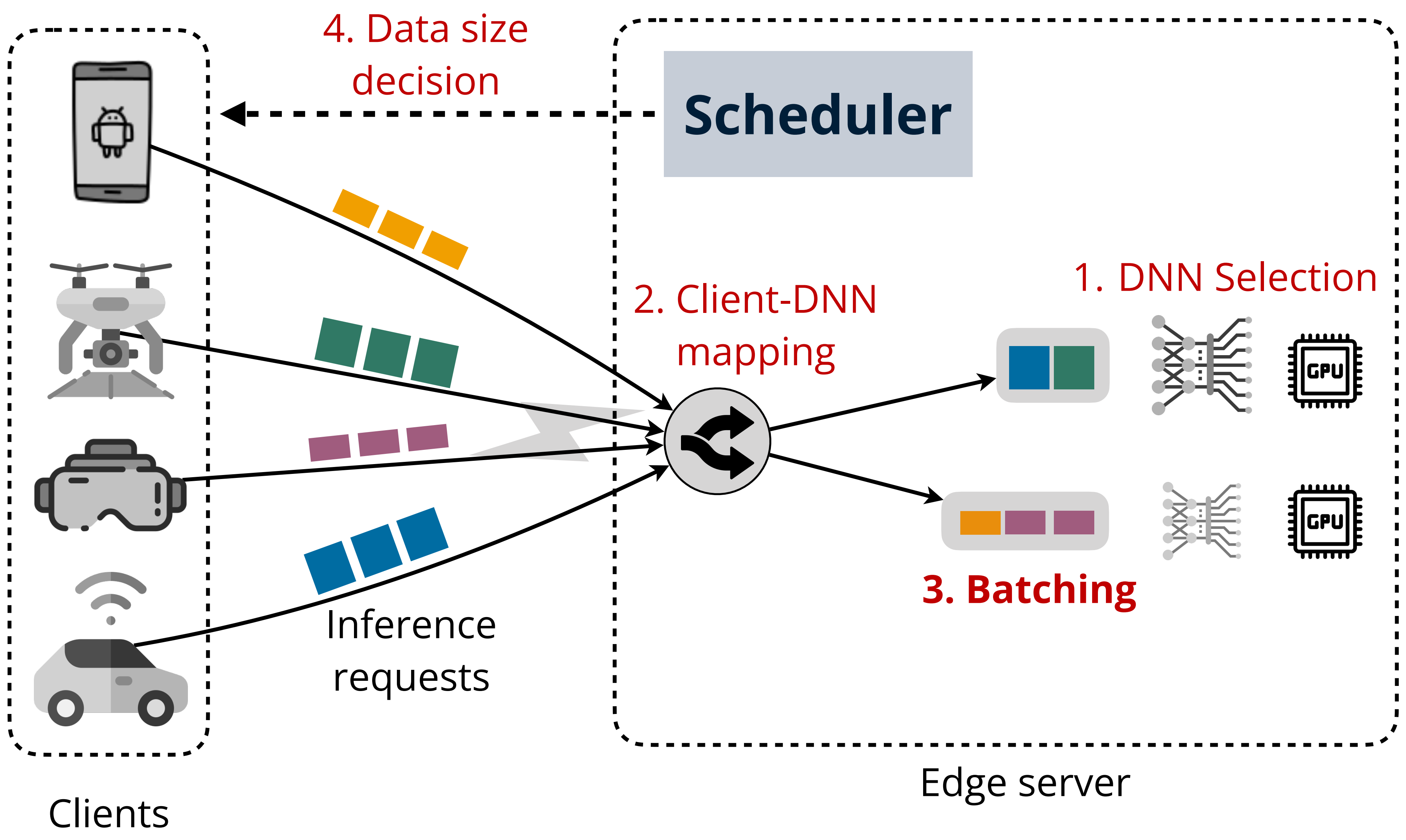
2. **Mapping every client (their requests) to the selected DNN variants**

3. Deciding the batch size of every DNN variant for serving multiple clients

4. Informing clients about their mapped DNN and data sizes

Solve continuously without violating end-to-end latency SLO

Jellyfish has to solve a complex scheduling problem

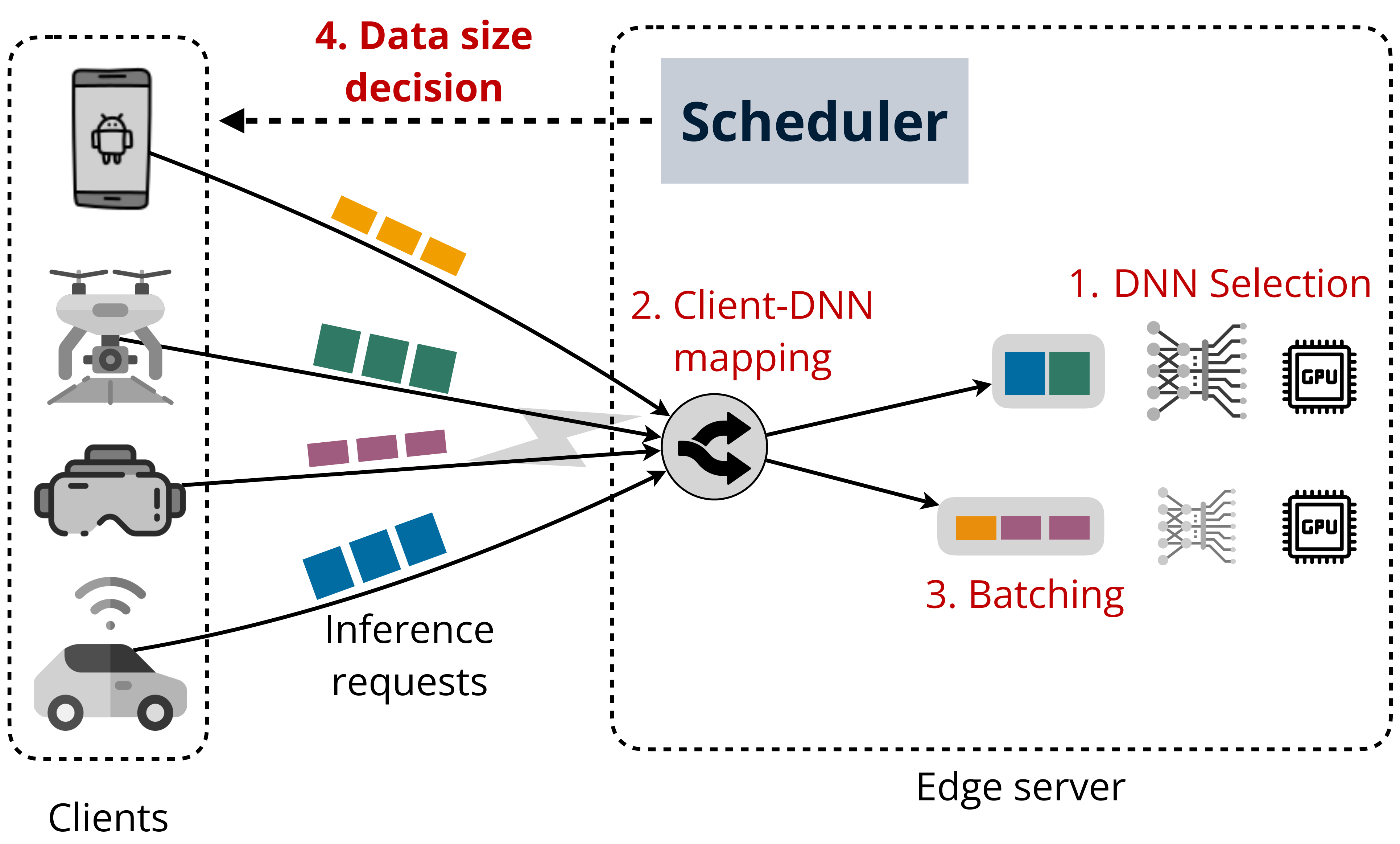


The scheduling problem involves **multiple complex steps**

1. Selection of a few DNN variants on a limited amount of compute resources
2. Mapping every client (their requests) to the selected DNN variants
3. **Deciding the batch size of every DNN variant for serving multiple clients**
4. Informing clients about their mapped DNN and data sizes

Solve continuously without violating end-to-end latency SLO

Jellyfish has to solve a complex scheduling problem



The scheduling problem involves **multiple complex steps**

1. Selection of a few DNN variants on a limited amount of compute resources
2. Mapping every client (their requests) to the selected DNN variants
3. Deciding the batch size of every DNN variant for serving multiple clients
4. **Informing clients about their mapped DNN and data sizes**

Solve continuously without violating end-to-end latency SLO

Formulate the problem as a mixed-integer linear program (MILP)

Maximize overall accuracy

$$\begin{aligned}
\max_{\{x,b\}} \quad & \boxed{\textstyle\sum_{i,j,k} a_j \cdot \lambda_i \cdot x_{ijk}} && (1)\\
\text{s.t.} \quad & \textstyle\sum_{j,k} x_{ijk} = 1, \forall i && (2)\\
& \textstyle\sum_{j} z_{kj} \leq 1, \forall k && (3)\\
& z_{kj} \geq x_{ijk}, \forall i, j, k && (4)\\
& \textstyle\sum_{j,k} x_{ijk} \cdot 2l_j(b_k) \leq \sum_{j,k} x_{ijk} \cdot L_{ij}, \forall i && (5)\\
& \textstyle\sum_{i,j} x_{ijk} \cdot \lambda_i \leq \sum_{j} z_{kj} \cdot t_j(b_k), \forall k && (6)\\
\text{vars} \quad & x_{ijk}, z_{kj} \in \{0,1\}, b_k \in [1 \ldots B]
\end{aligned}$$

Satisfy latency & throughput constraints

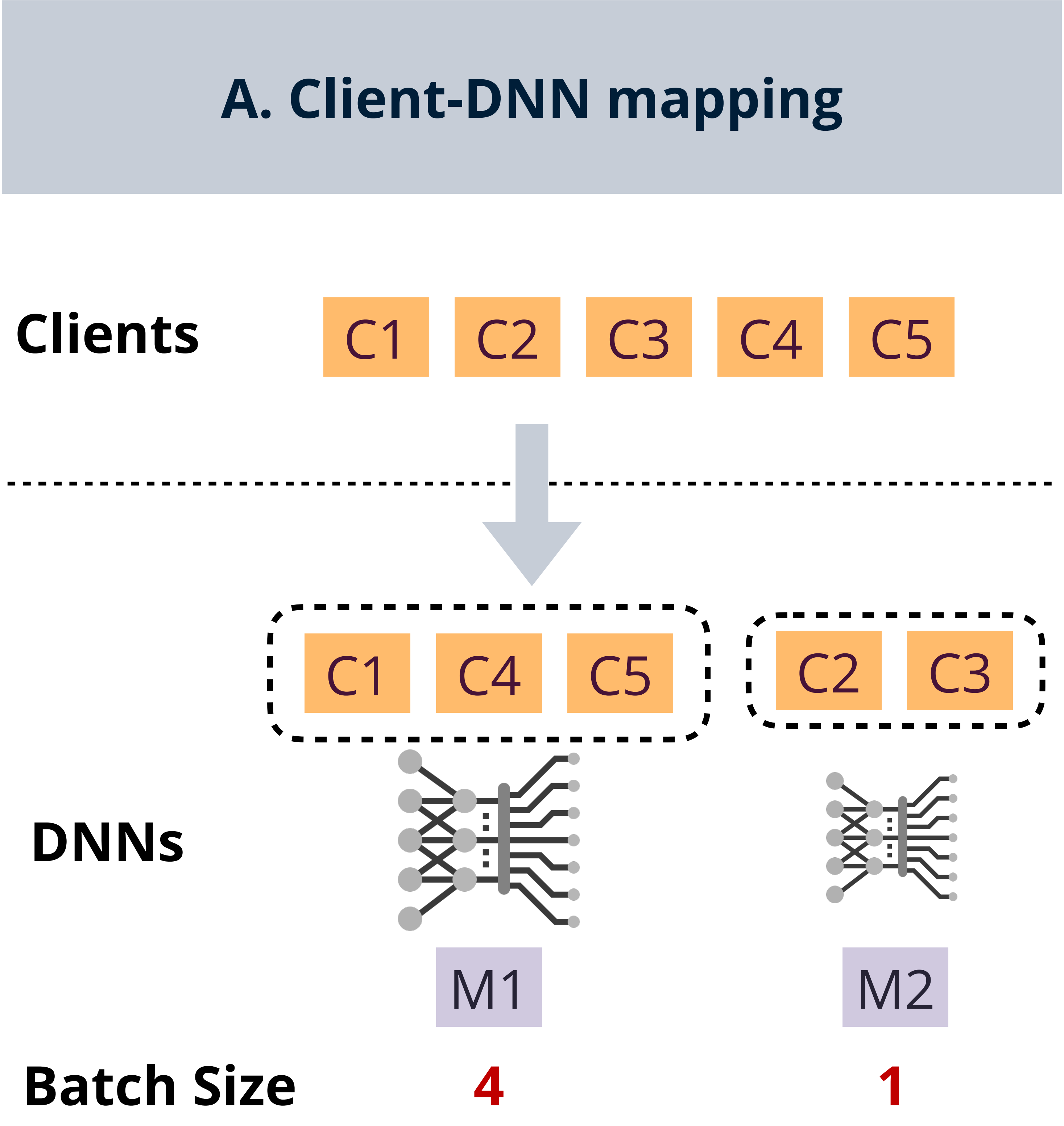
Not feasible to run in real-time (sub-seconds)

Existing MILP solvers take around
20 seconds to 15 minutes

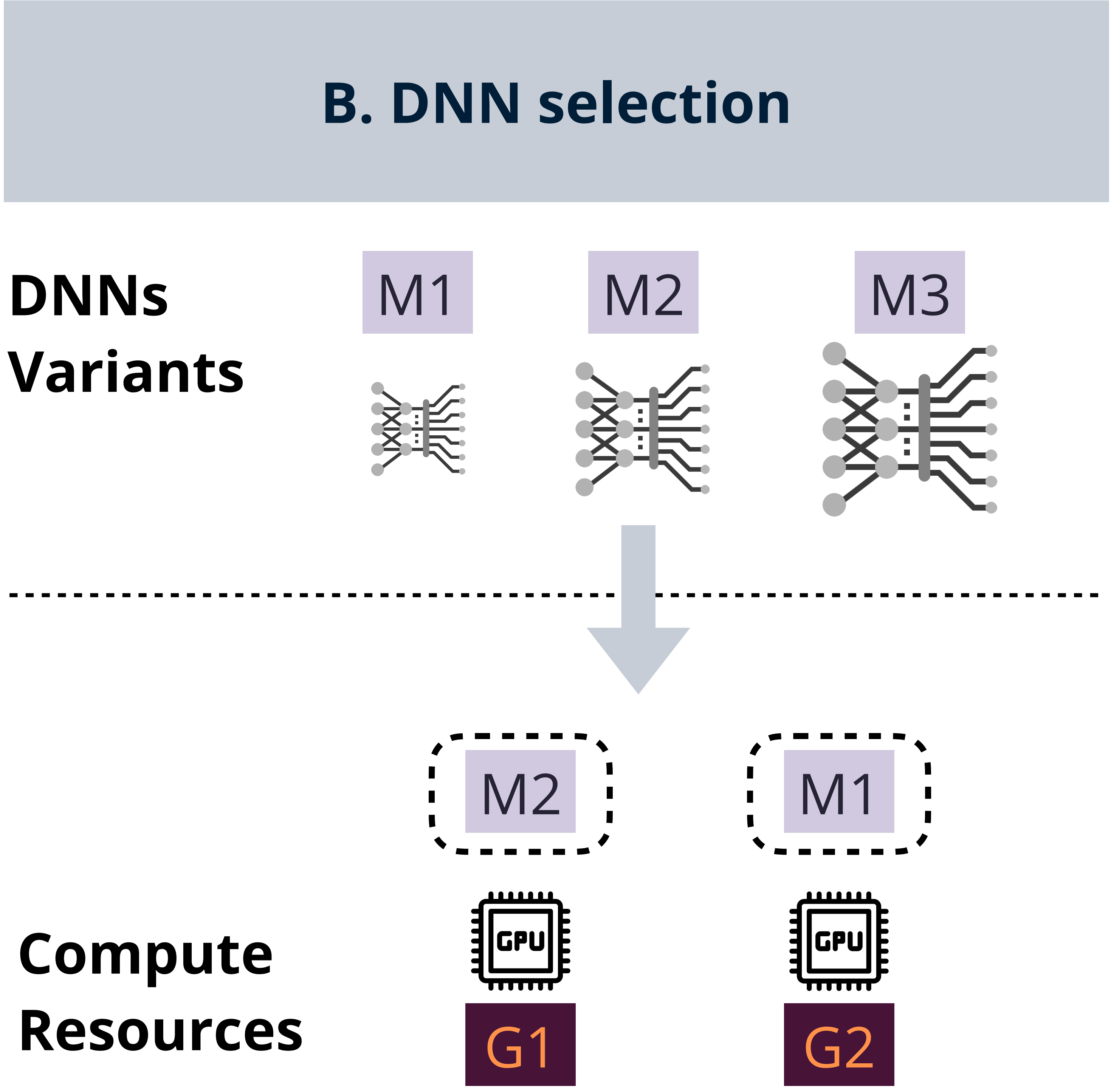
With 4 threads, 4 GPU workers, 16 DNNs, 16 Clients, and batch size 12

How to solve the scheduling problem continuously in real-time?

Jellyfish decomposes the problem into two sub-problems



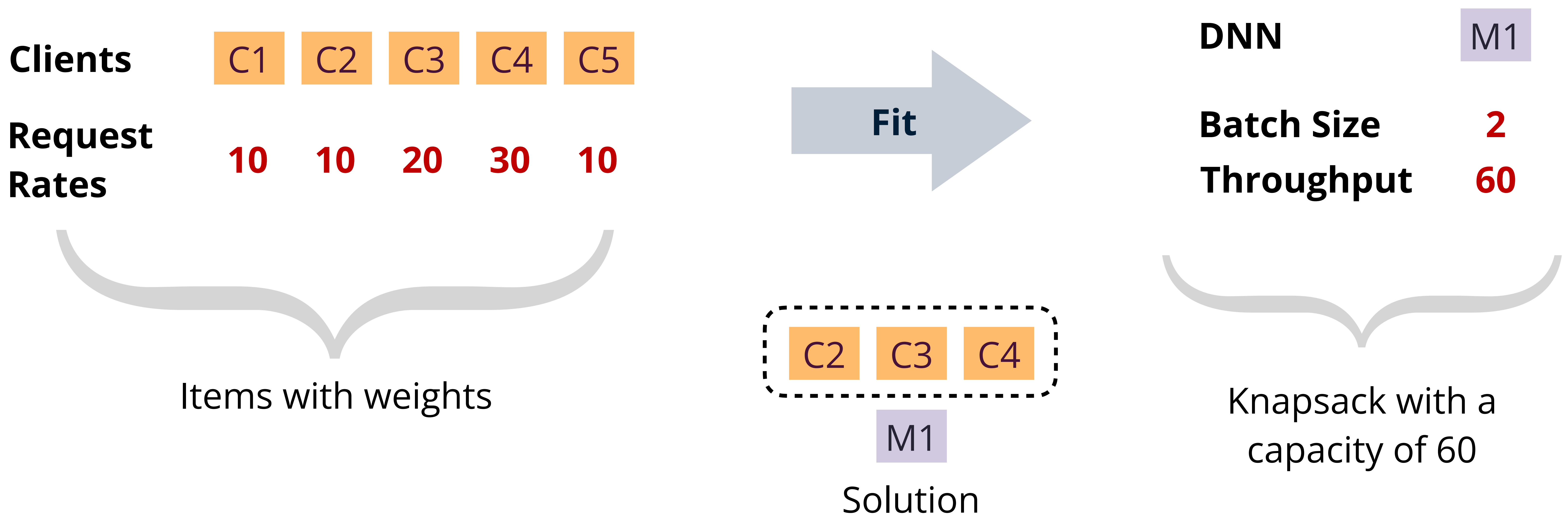
- Optimize accuracy
- Satisfy latency & throughput constraints



- Optimize accuracy
- Serve a maximum number of requests

A. Client-DNN mapping

As a standard 0-1 knapsack problem



But we have to solve the standard knapsack problem for every batch size

A. Client-DNN mapping

One-shot dynamic programming to solve for all batch sizes in one go

DNN	M1	
Batch Size	**2**	**3**
Throughput	**60**	**80**

Clients	C1	C2	C3	C4	C5
Request Rates	**10**	**10**	**20**	**30**	**10**

One-shot DP

Batch Size

2 **3**

DNN throughput

Sorted clients

	0	10	20	30	40	50	60	70	80
0	0	0	0	0	0	0	0	0	0
C1	0	10	10	10	10	10	10	10	10
C2	0	10	20	20	20	20	20	20	20
C3	0	10	20	30	**40**	40	40	40	40
C4	0	10	20	30	40	50	$\boxed{60}$	0	0
C5	0	10	20	30	40	50	60	0	0

Clients that violate compute budget constraints

- To **optimize accuracy**, we first map clients on a bigger DNN and then the remaining clients on smaller DNNs

A. Client-DNN mapping

One-shot dynamic programming to solve for all batch sizes in one go

DNN	M1	
Batch Size	**2**	**3**
Throughput	**60**	**80**

Clients	C1	C2	C3	C4	C5
Request Rates	**10**	**10**	**20**	**30**	**10**

One-shot DP

Batch Size

2 **3**

DNN throughput

Sorted clients

	0	10	20	30	40	50	60	70	80
0	0	0	0	0	0	0	0	0	0
C1	0	10	10	10	10	10	10	10	10
C2	0	10	20	20	20	20	20	20	20
C3	0	10	20	30	**40**	40	40	40	40
C4	0	10	20	30	40	50	**60**	0	0
C5	0	10	20	30	40	50	60	0	0

Clients that violate compute budget constraints

- To **optimize accuracy**, we first map clients on a bigger DNN and then the remaining clients on smaller DNNs

A. Client-DNN mapping

One-shot dynamic programming to solve for all batch sizes in one go

DNN	M1				
Batch Size	**2**	**3**			
Throughput	**60**	**80**			
Clients	C1	C2	C3	C4	C5
Request Rates	**10**	**10**	**20**	**30**	**10**

One-shot DP

Batch Size

2 **3**

DNN throughput

Sorted clients

	0	10	20	30	40	50	60	70	80
0	0	0	0	0	0	0	0	0	0
C1	0	10	10	10	10	10	10	10	10
C2	0	10	20	20	20	20	20	20	20
C3	0	10	20	30	**40**	40	40	40	40
C4	0	10	20	30	40	50	**60**	0	0
C5	0	10	20	30	40	50	60	0	0

Clients that violate compute budget constraints

- To **optimize accuracy**, we first map clients on a bigger DNN and then the remaining clients on smaller DNNs

A. Client-DNN mapping

One-shot dynamic programming to solve for all batch sizes in one go

DNN	M1	
Batch Size	**2**	**3**
Throughput	**60**	**80**

Clients	C1	C2	C3	C4	C5
Request Rates	**10**	**10**	**20**	**30**	**10**

One-shot DP

Batch Size

2 **3**

DNN throughput

Sorted clients

	0	10	20	30	40	50	60	70	80
0	0	0	0	0	0	0	0	0	0
C1	0	10	10	10	10	10	10	10	10
C2	0	10	20	20	20	20	20	20	20
C3	0	10	20	30	**40**	40	40	40	40
C4	0	10	20	30	40	50	**60**	0	0
C5	0	10	20	30	40	50	60	0	0

Clients that violate compute budget constraints

- To **optimize accuracy**, we first map clients on a bigger DNN and then the remaining clients on smaller DNNs

B. DNN selection

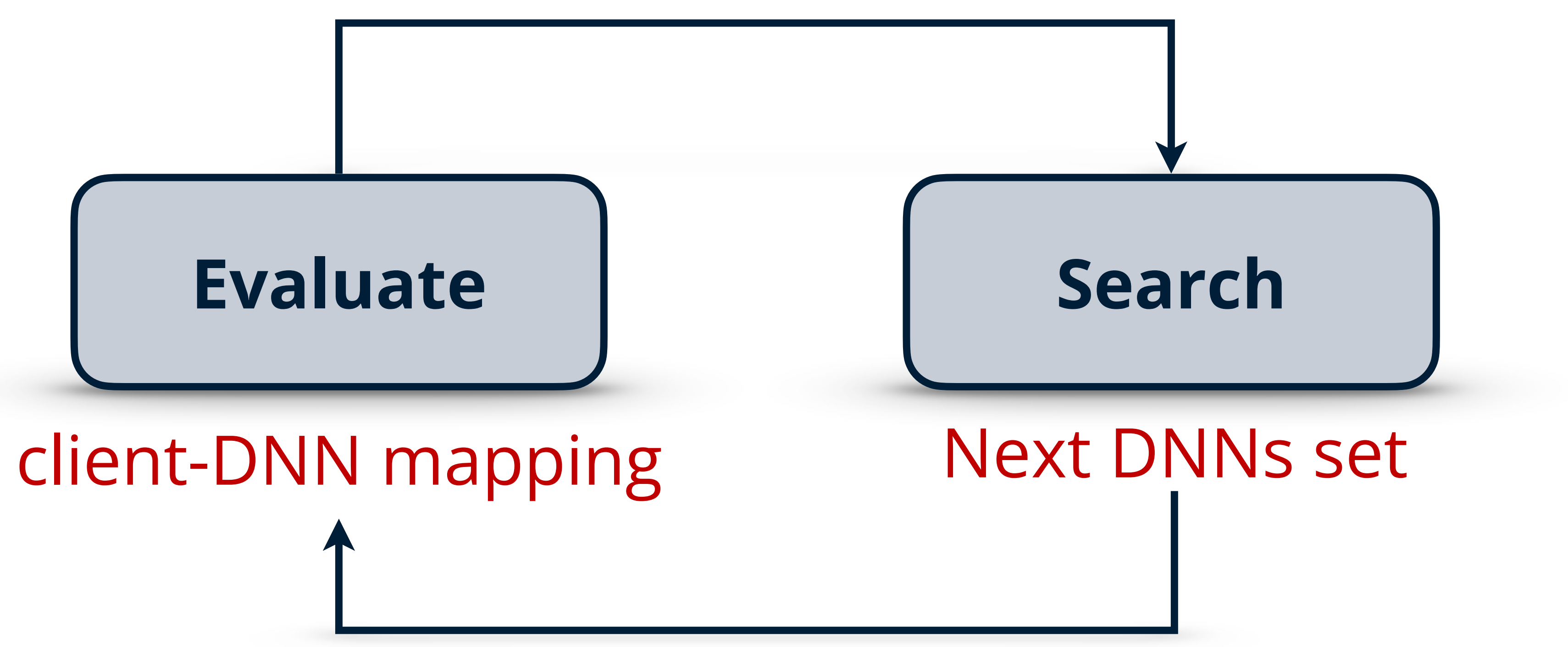
An iterative search process

- Exhaustively searching for the DNN set from all possible combinations of DNN variants can become **expensive**

Searching **r** DNN instances from a DNNs zoo with **n** DNN variants is **combinatorial**: $\binom{n + r - 1}{r}$

- An iterative search process that uses the **client-DNN mapping** to **evaluate** DNN sets

Iterative process



- **Simulated annealing (SA)** to search for the next set of DNN instances

How well does Jellyfish perform?

Experimental setup

Jellyfish is evaluated on a popular **video analytics task** and **real-world network traces**

- **Task:** vehicle object detection
- **Videos:** three traffic videos 10min each

Metrics

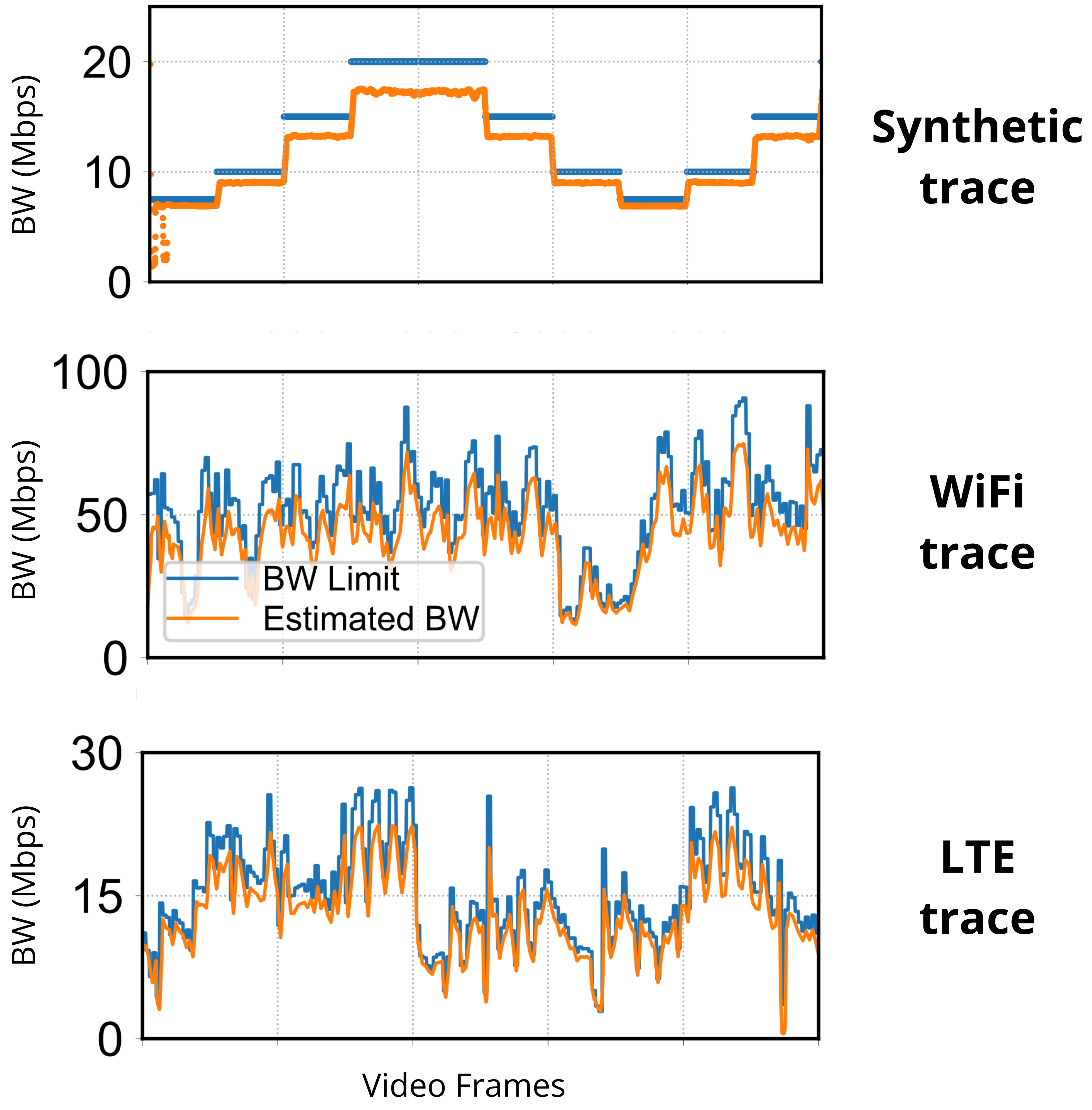
- **Analytics accuracy:** standard F1 score
- **Miss rate:** latency SLO violations

Clients Configuration

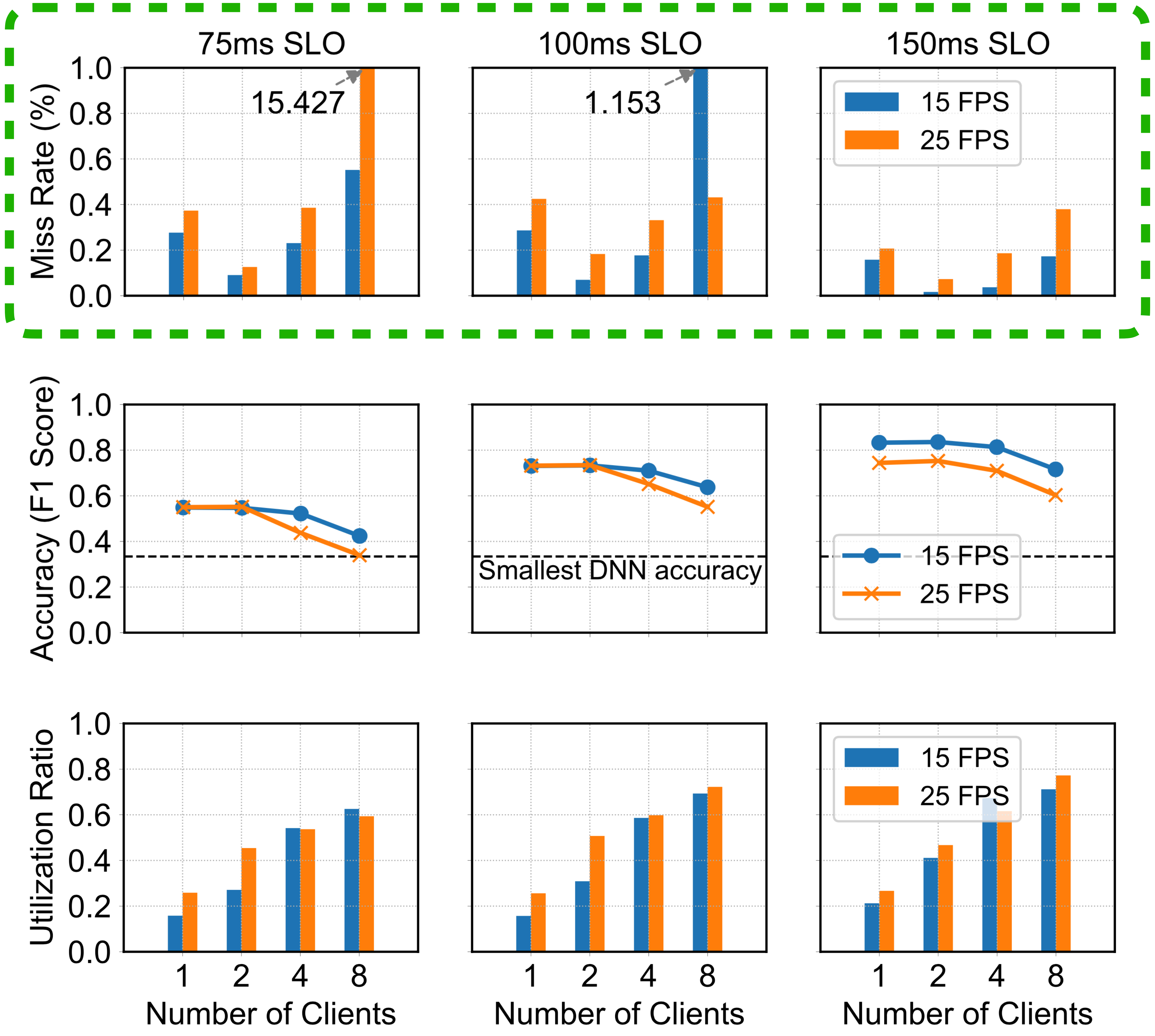
- **Number of clients:** {1, 2, 4, 8}
- **SLOs:** {75, 100, 150} milliseconds (ms)
- **FPS:** {15, 25}

Server Configuration

- **GPUs:** 2 RTX2080Ti
- **DNNs:** 16 YOLOv4 variants

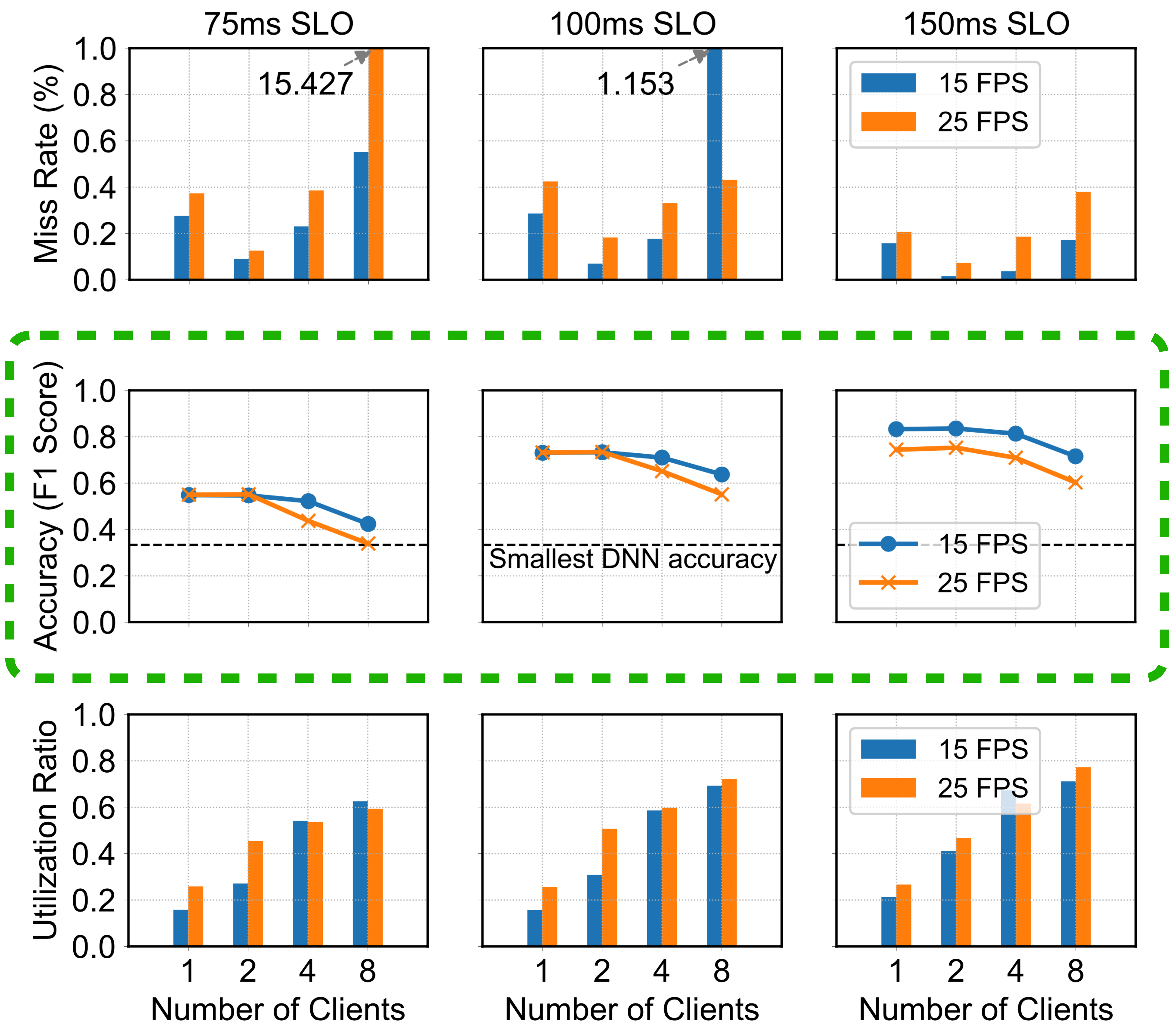


End-to-end performance on synthetic network trace



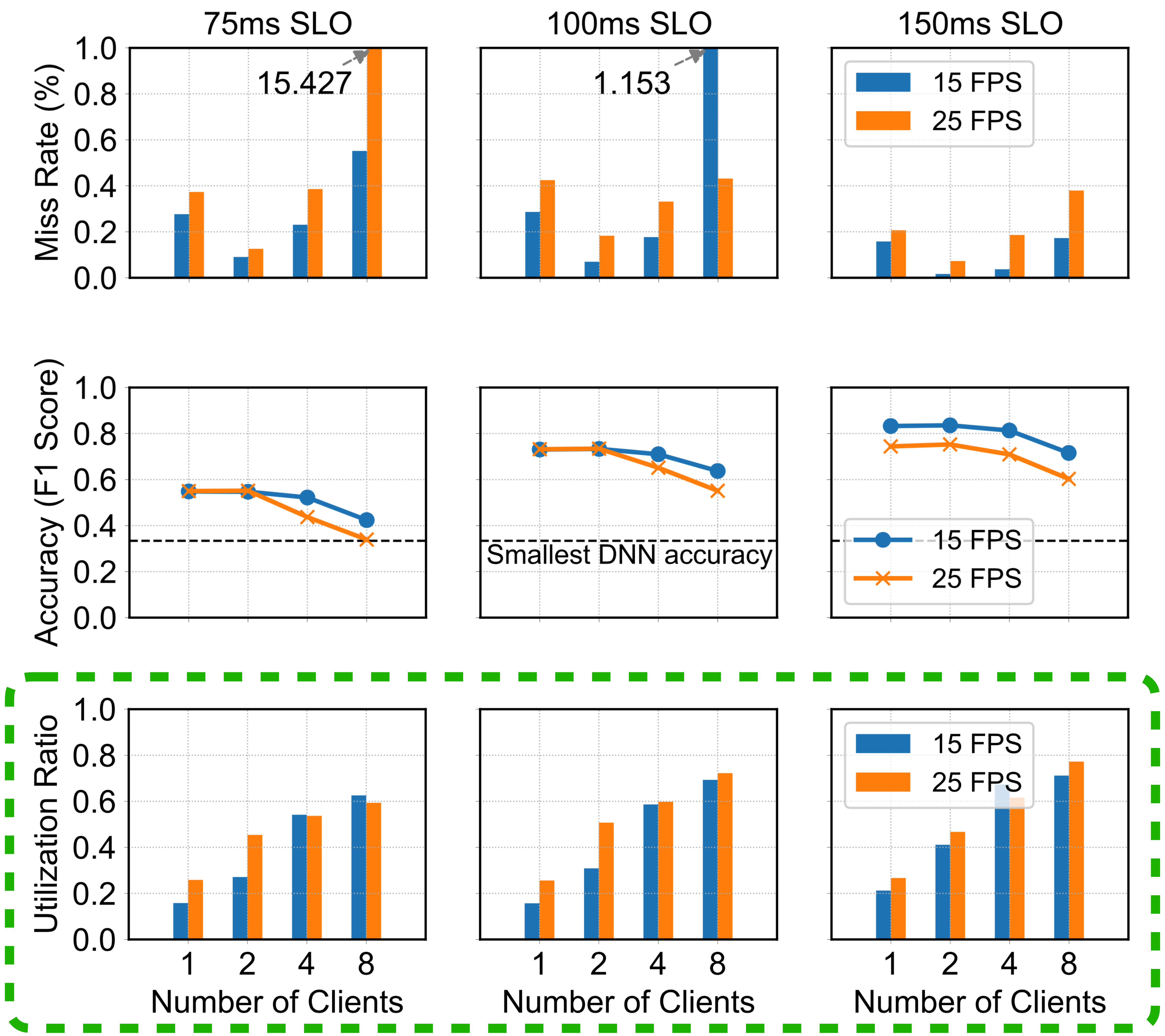
- **Achieves extremely low miss rates (≤1%) when the system is not overloaded**
- Maintains high accuracy by selecting bigger DNNs whenever possible
- Maintains high worker utilization (up to 75%) when the system becomes more saturated

End-to-end performance on synthetic network trace



- Achieves extremely low miss rates (≤1%) when the system is not overloaded
- **Maintains high accuracy by selecting bigger DNNs whenever possible**
- Maintains high worker utilization (up to 75%) when the system becomes more saturated

End-to-end performance on synthetic network trace



- Achieves extremely low miss rates (≤1%) when the system is not overloaded
- Maintains high accuracy by selecting bigger DNNs whenever possible
- **Maintains high worker utilization (up to 75%) when the system becomes more saturated**

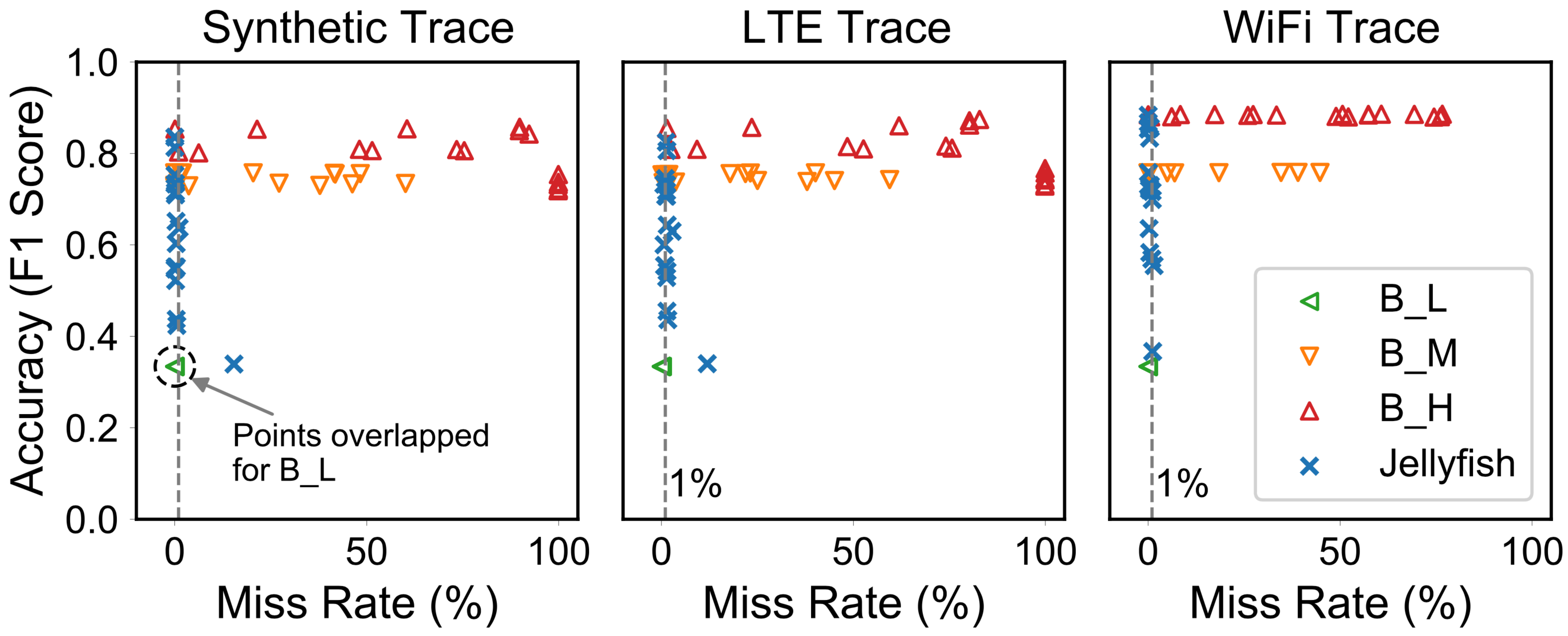
Comparison with baselines on three network traces

Server:

- **Scheduler:** EDF-like **[Clockwork, OSDI'20]**
- **Three baseline variants:** lowest DNN (B_L), middle DNN (B_M), and biggest DNN (B_H)

Client:

- **Data adaptation:** Bandwidth-aware **[AWStream, SIGCOMM'18]**

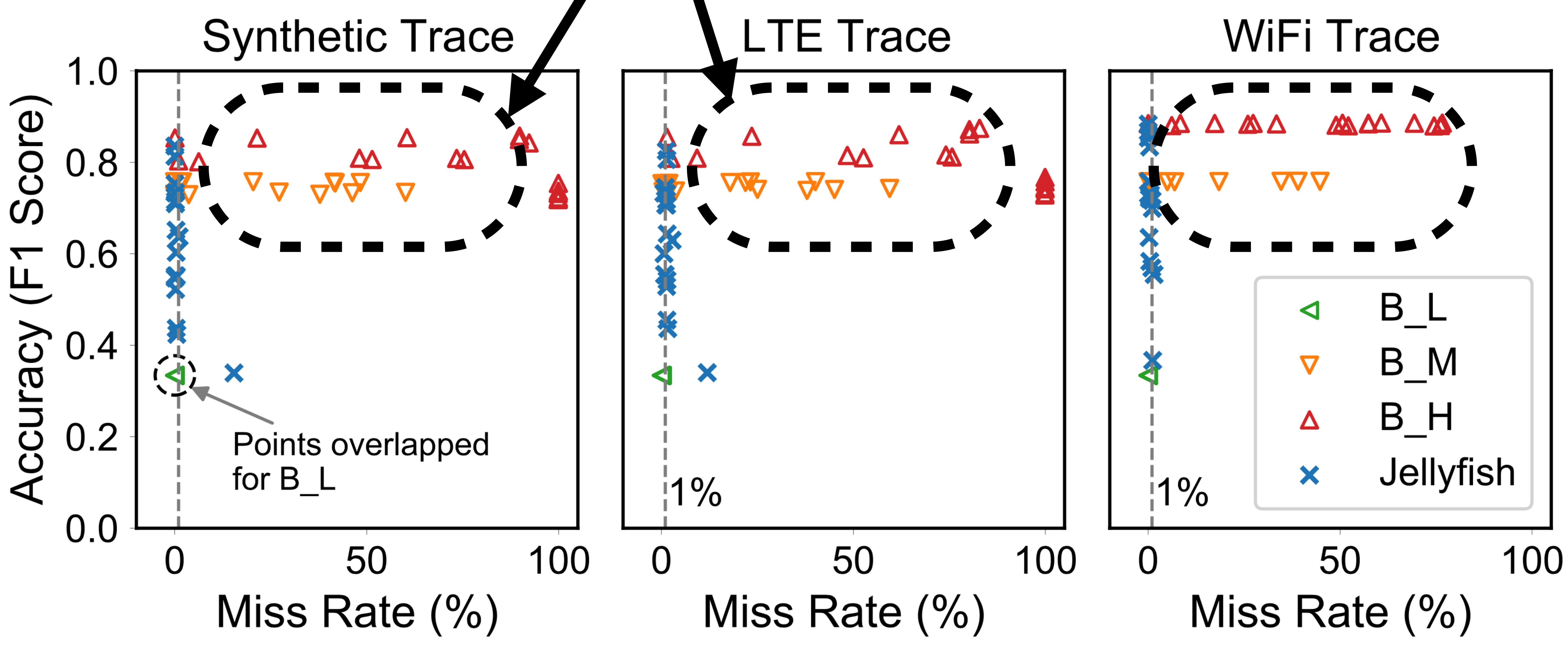


Comparison with baselines on three network traces

Server:
- **Scheduler:** EDF-like **[Clockwork, OSDI'20]**
- **Three baseline variants:** lowest DNN (B_L), middle DNN (B_M), and biggest DNN (B_H)

Client:
- **Data adaptation:** Bandwidth-aware **[AWStream, SIGCOMM'18]**

- **Baselines with bigger static DNNs have higher miss rates**

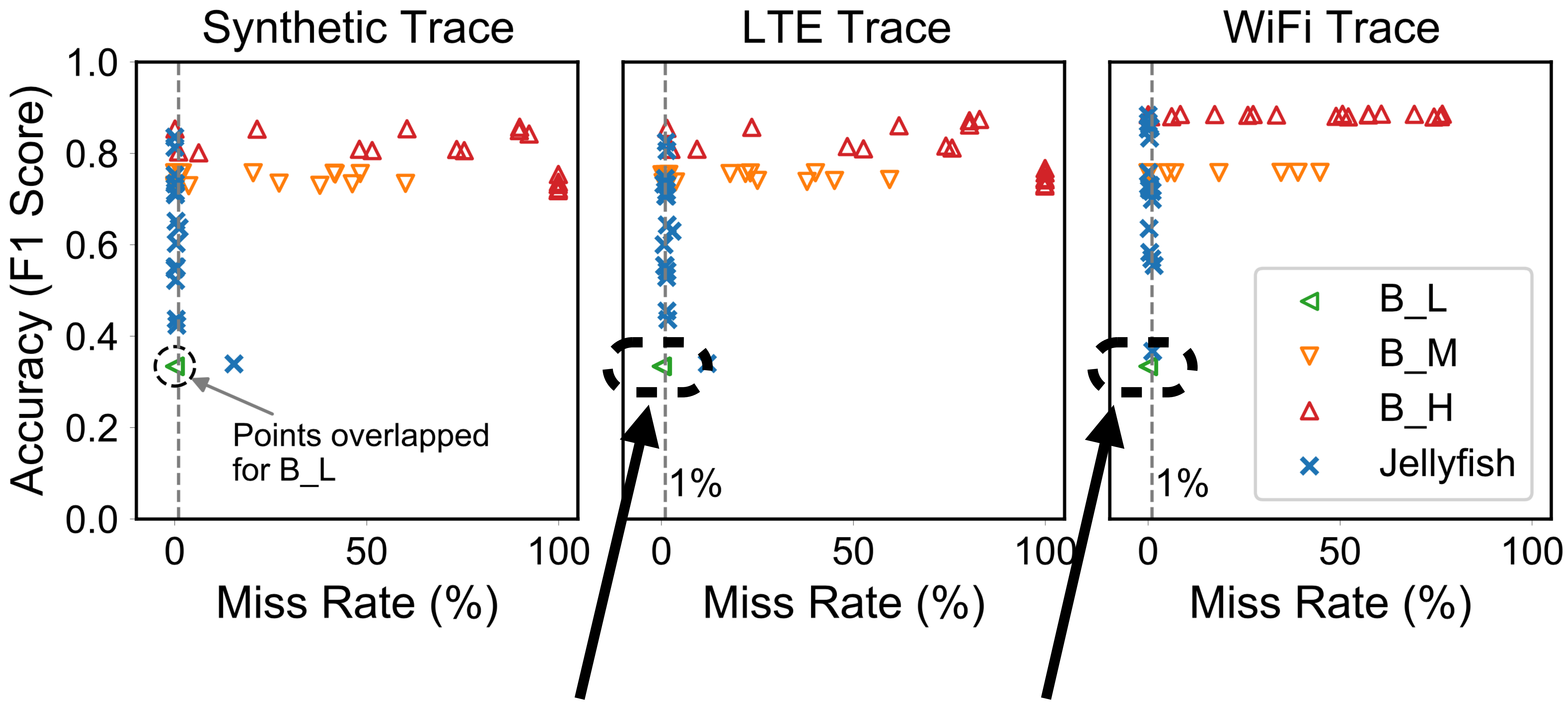


Comparison with baselines on three network traces

Server:
- **Scheduler:** EDF-like **[Clockwork, OSDI'20]**
- **Three baseline variants:** lowest DNN (B_L), middle DNN (B_M), and biggest DNN (B_H)

Client:
- **Data adaptation:** Bandwidth-aware **[AWStream, SIGCOMM'18]**

- Baselines with bigger static DNNs have higher miss rates



- **Baselines with smaller static DNNs have lower miss rates but also lower accuracy**

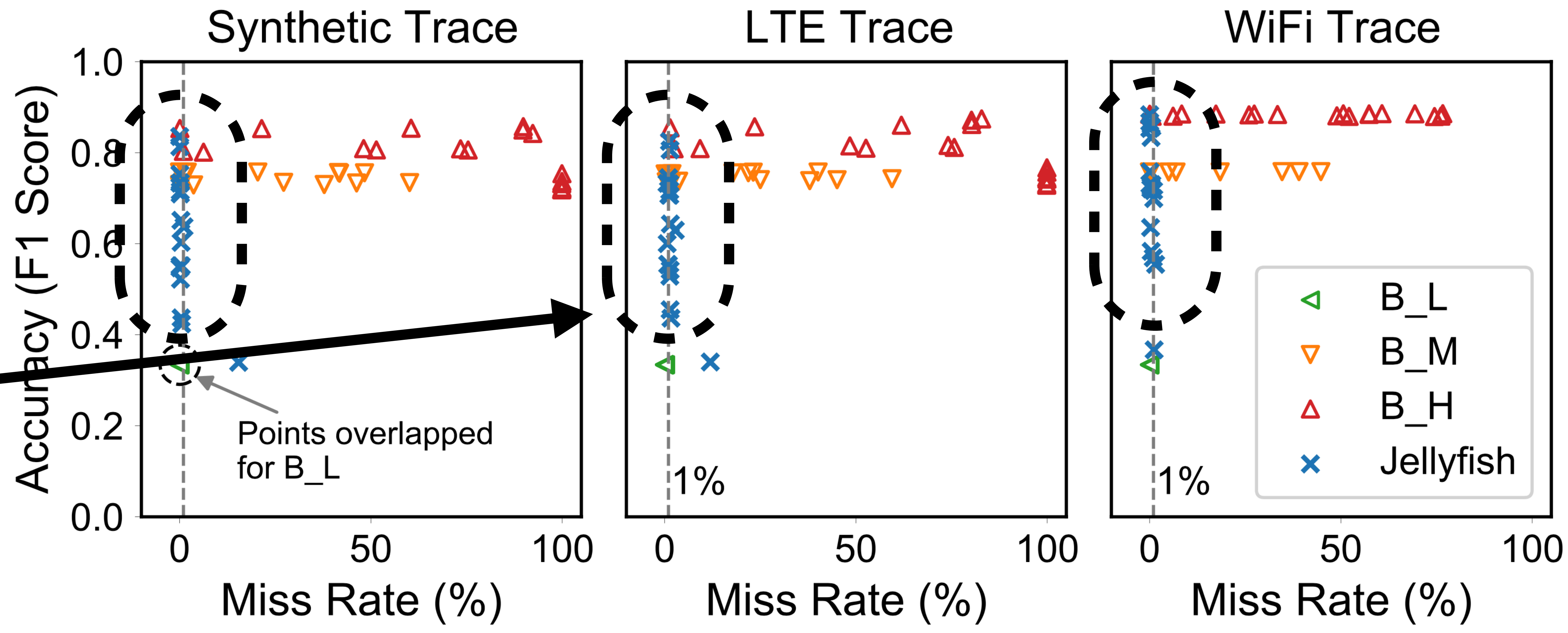
Comparison with baselines on three network traces

Server:
- **Scheduler:** EDF-like **[Clockwork, OSDI'20]**
- **Three baseline variants:** lowest DNN (B_L), middle DNN (B_M), and biggest DNN (B_H)

Client:
- **Data adaptation:** Bandwidth-aware **[AWStream, SIGCOMM'18]**

- Baselines with bigger static DNNs have higher miss rates

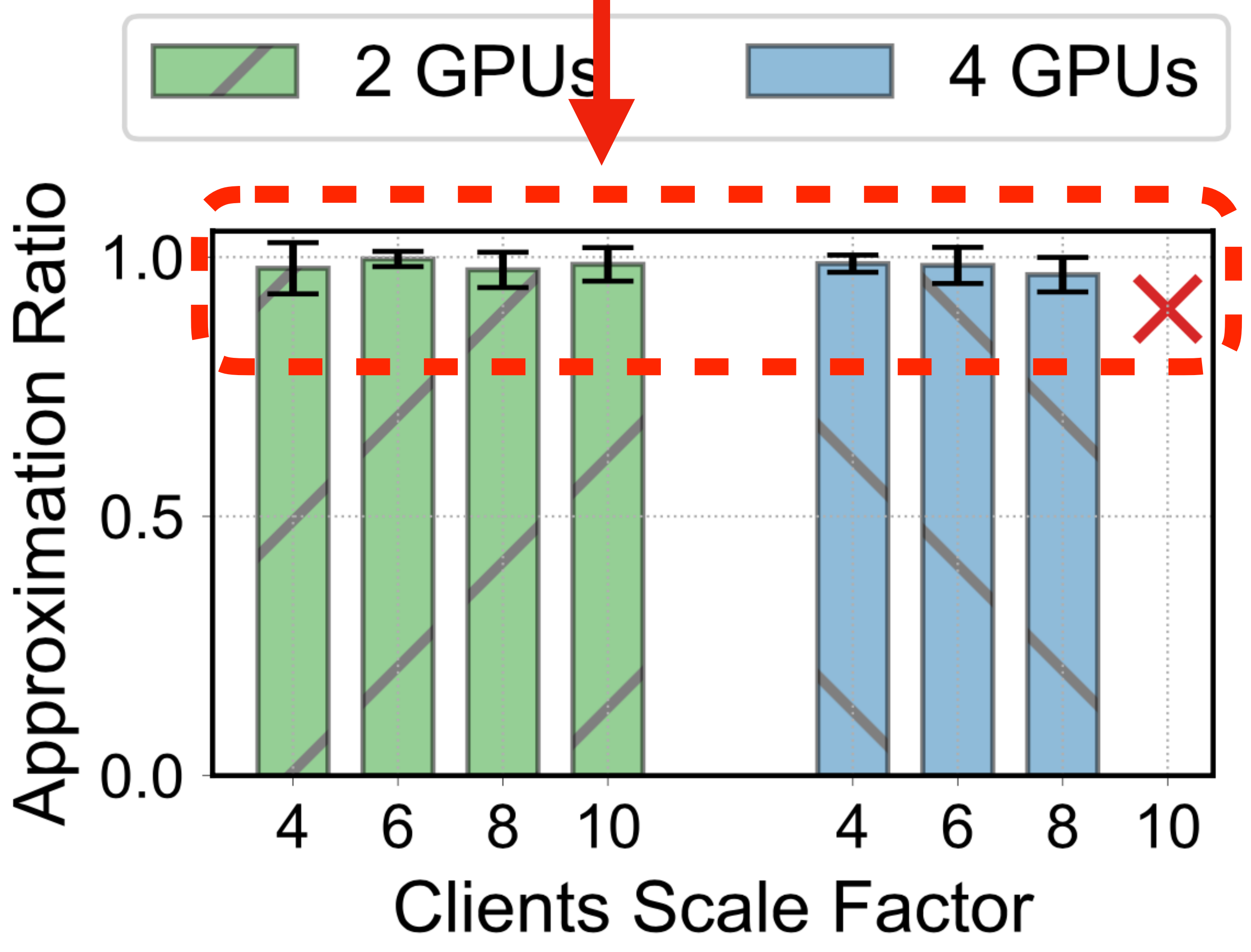
- **Jellyfish adaptively selects optimal DNNs and thus achieves low miss rates while maintaining high accuracy**



- Baselines with smaller static DNNs have lower miss rates but also lower accuracy

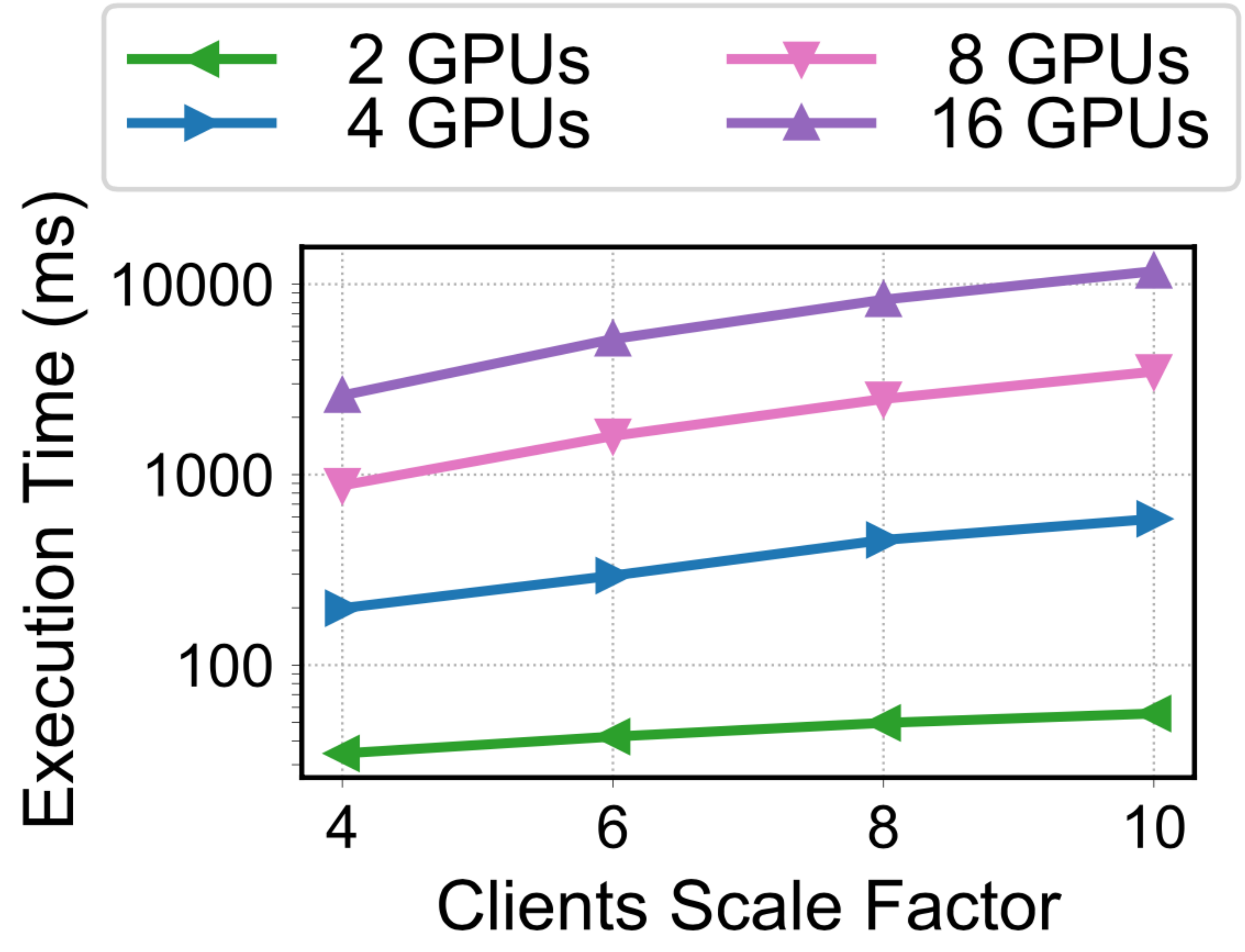
Jellyfish scheduler is near-optimal and runs in real-time

- **The approximation ratio compared to MILP ranges from 0.966 to 0.996**



(a) Approximation ratio (mean)

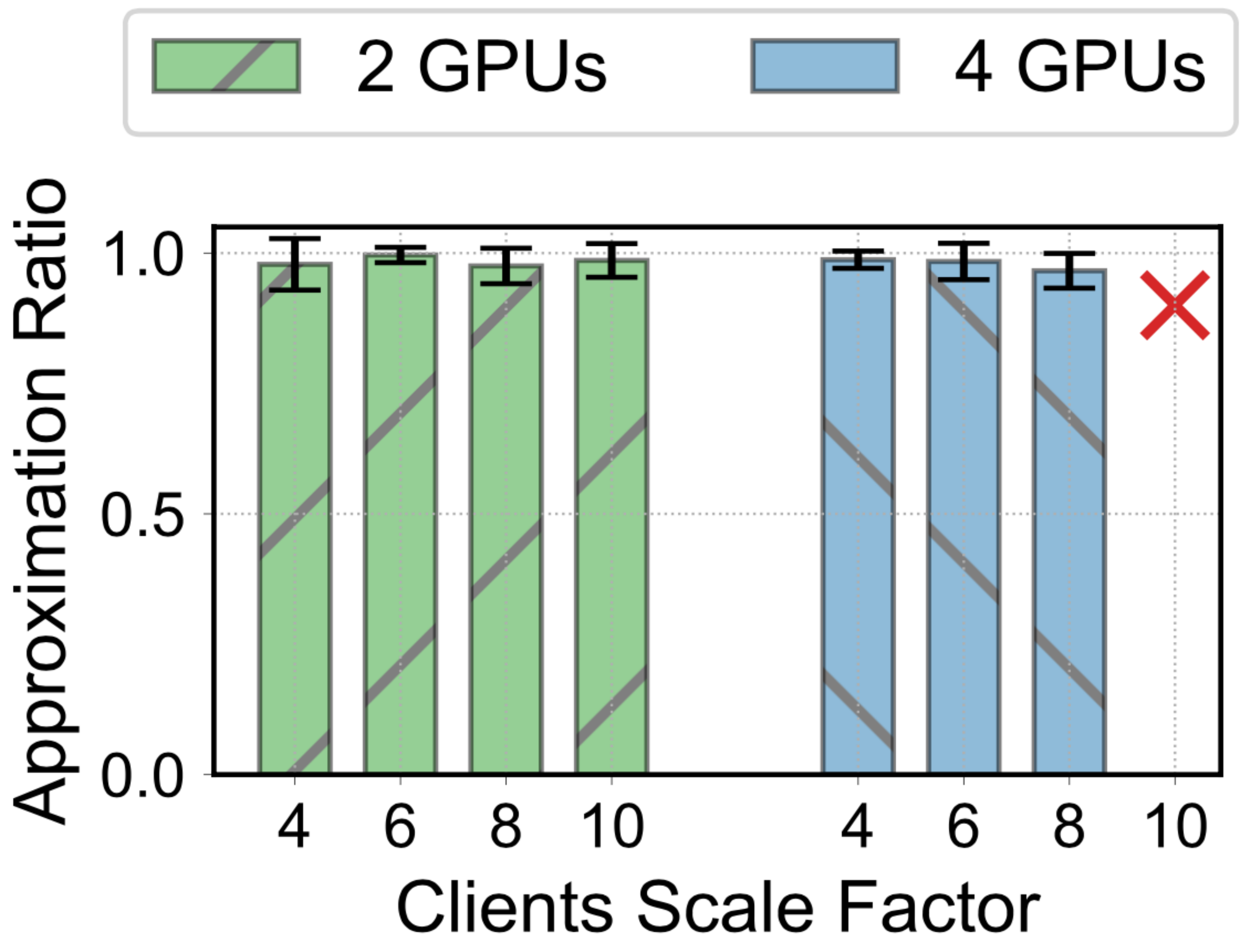
- For up to 8 GPUs and 32 clients, the scheduler has running times less than seconds



(b) Execution time in log scale

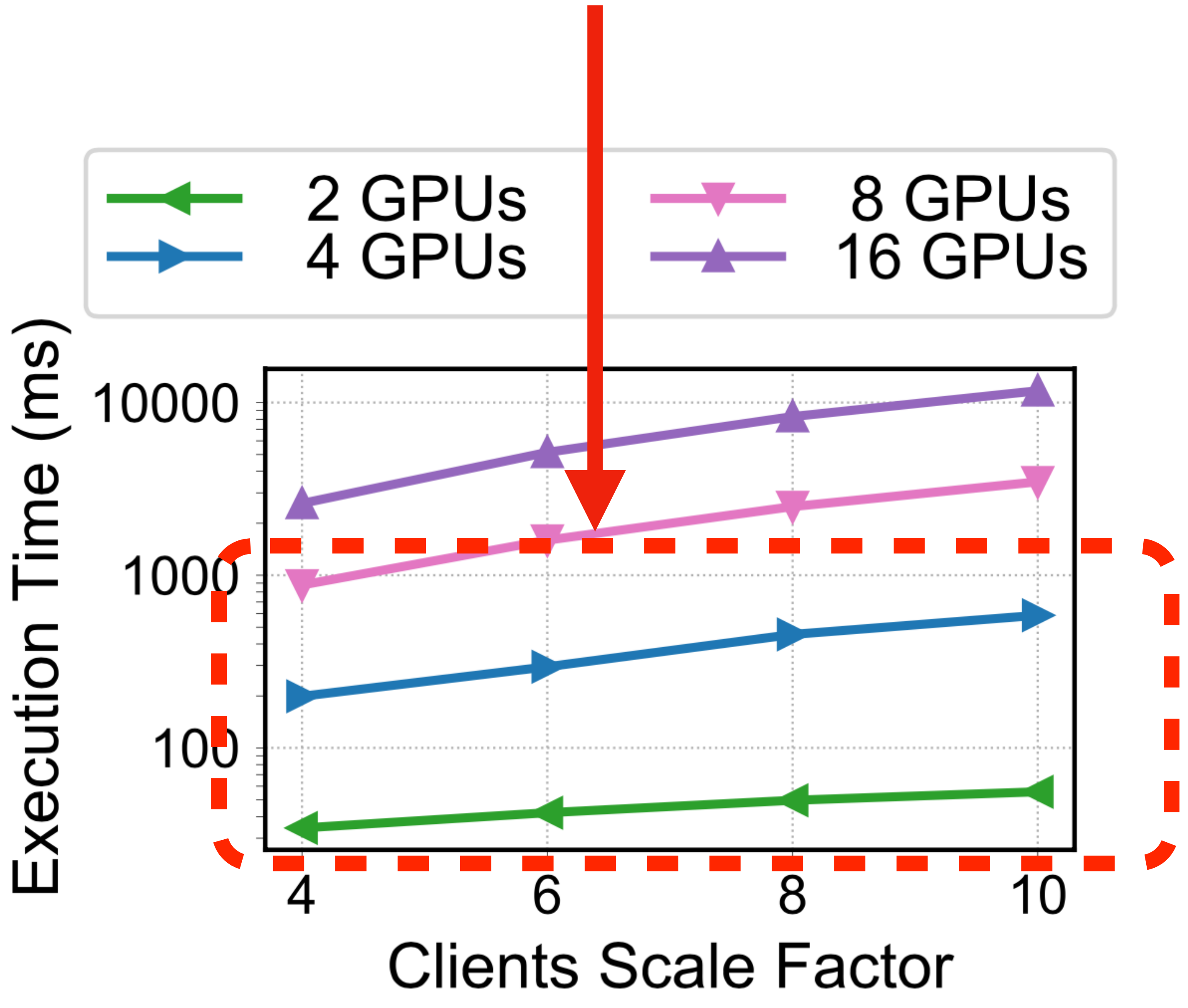
Jellyfish scheduler is near-optimal and runs in real-time

- The approximation ratio compared to MILP ranges from 0.966 to 0.996



(a) Approximation ratio (mean)

- **For up to 8 GPUs and 32 clients, the scheduler has running times less than seconds**



(b) Execution time in log scale

Discussion and future work

- **Request rate adaptation** is not incorporated in the current version
- **Compute budget estimation** depends on the accurate estimation of compressed data size, which is difficult due to the changing data content
- The system must be tuned for stable performance (i.e., for **predictability**)

Summary

- **Timely inference serving** over dynamic edge networks is important and challenging
- We propose **Jellyfish** which…
  - aims to fulfill **end-to-end latency SLOs** specified over the variable network time and DNN inference time
  - employs **data and DNN adaptation jointly** and coordinates adaptation decisions for multiple clients
  - achieves extremely **low latency SLO violations** while maintaining **high accuracy**

Contact: Vinod Nigade

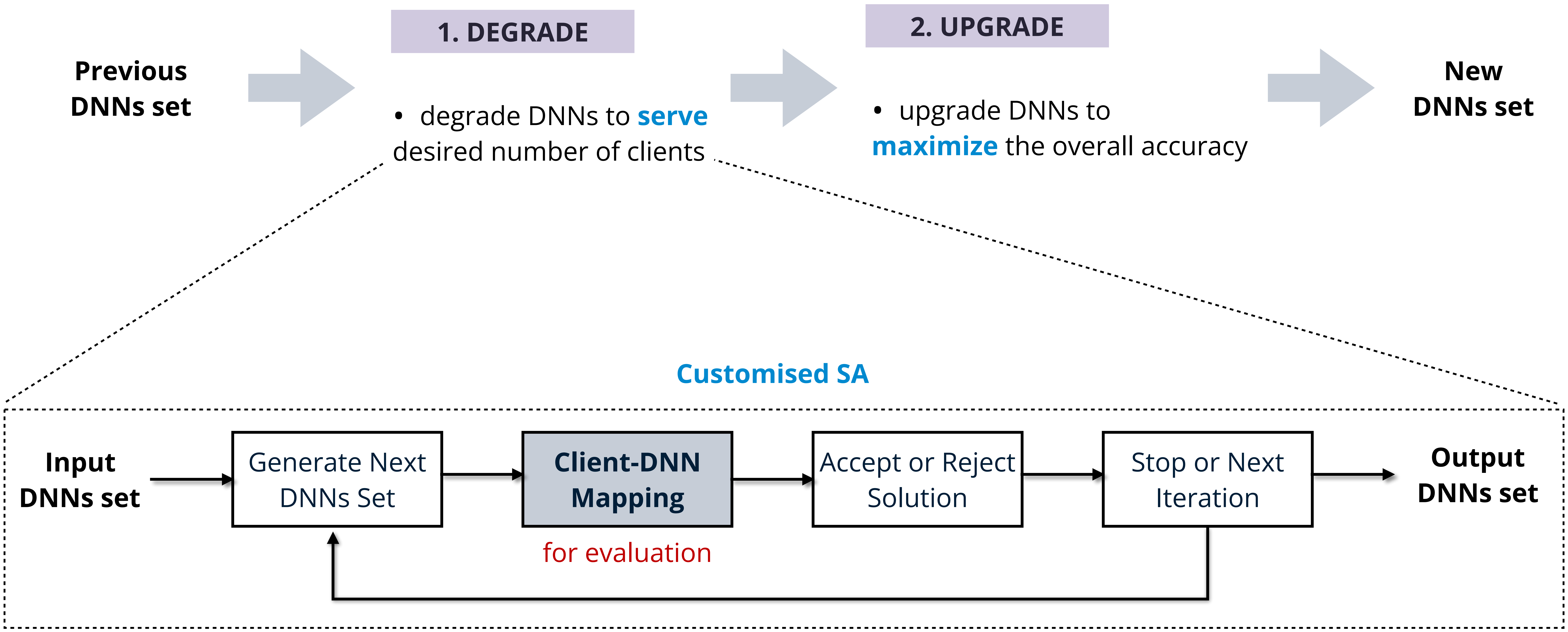
Email id: v.v.nigade@vu.nl

Source code: https://github.com/vuhpdc/jellyfish

Extra slides

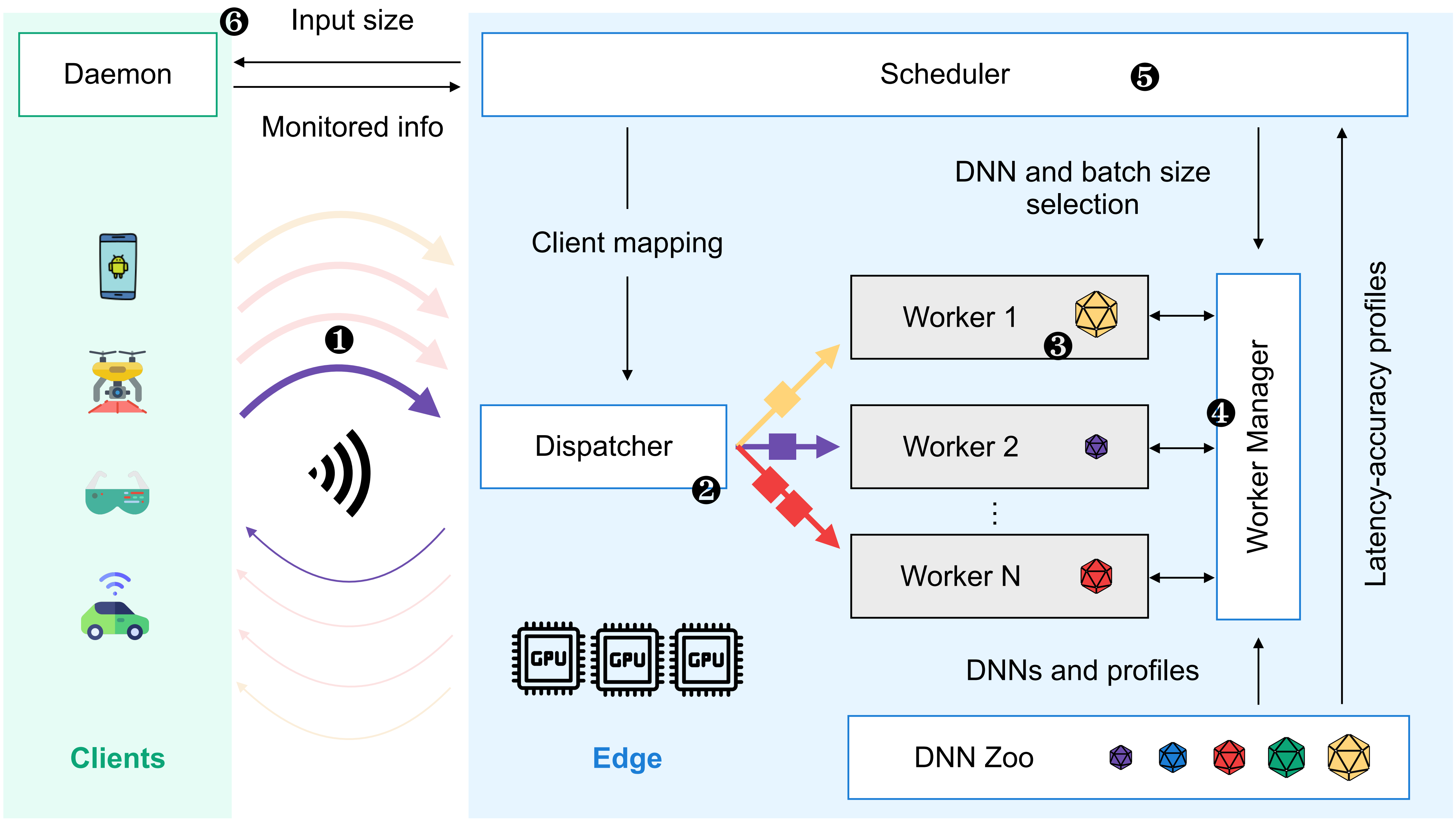
DNN selection

Unlike conventional SA, Jellyfish has **two sequential modes** of operation



More details in the paper

- DNN pre-fetching technique to minimize DNNs switching cost
- Client's bandwidth estimation
- System design



Jellyfish

Comparison to independently running data and DNN adaptation

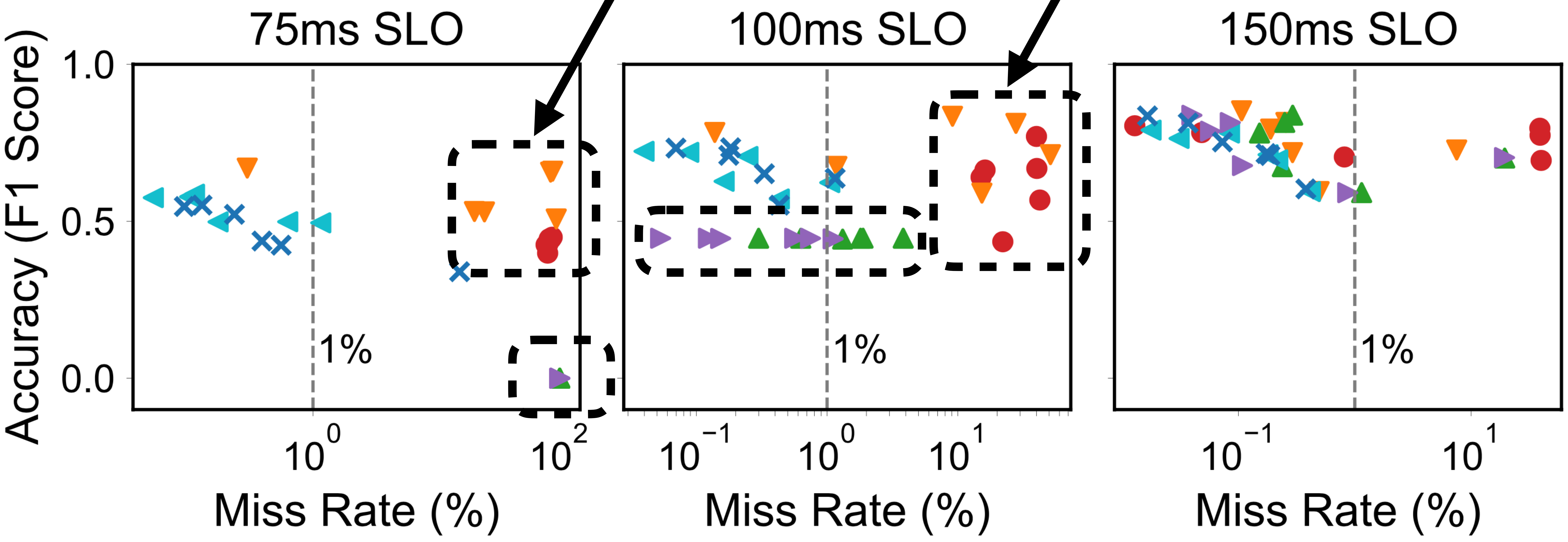
Data adaptation

- **DA_{off}:** disabled
- **DA_{bw}:** bandwidth-aware
- **DA_{slo}:** bandwidth and slo-aware

DNN adaptation:

- **$CB_{50\%}$:** 50% of the SLO as compute budget
- **$CB_{75\%}$:** 75% of the SLO as compute budget

- **Without proper coordination and alignment between data and DNN adaptation, we see high miss rates or low accuracy**



Performance on a large-scale setup with LTE trace

Clients Configuration

- **Number of clients:** {8, 16, 24, 32}
- **SLOs:** {100, 150} milliseconds (ms)
- **FPS:** 15
- **AWS instance:** t3.2xlarge

Server Configuration

- **GPUs:** 8 distributed NVIDIA T4
- **Worker AWS instance:** g4dn.2xlarge
- **Dispatcher & scheduler AWS instance:** c5.9xlarge

- **Jellyfish achieves miss rates within the acceptance range (1-3%), even on a large-scale setup with unstable inference timings**