COSTREAM: Learned Cost Models for Operator Placement in Edge-Cloud Environments

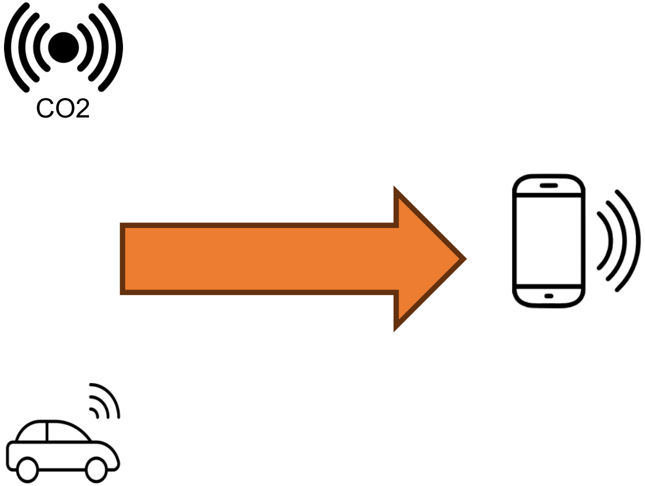
40th IEEE International Conference on Data Engineering
14th May 2024

Roman Heinrich[1,2*], Carsten Binnig[1,2], Harald Kornmayer[3], Manisha Luthra[1,2]
[1] DFKI Darmstadt, [2] TU Darmstadt, [3] DHBW Mannheim

* This work was done at DHBW Mannheim

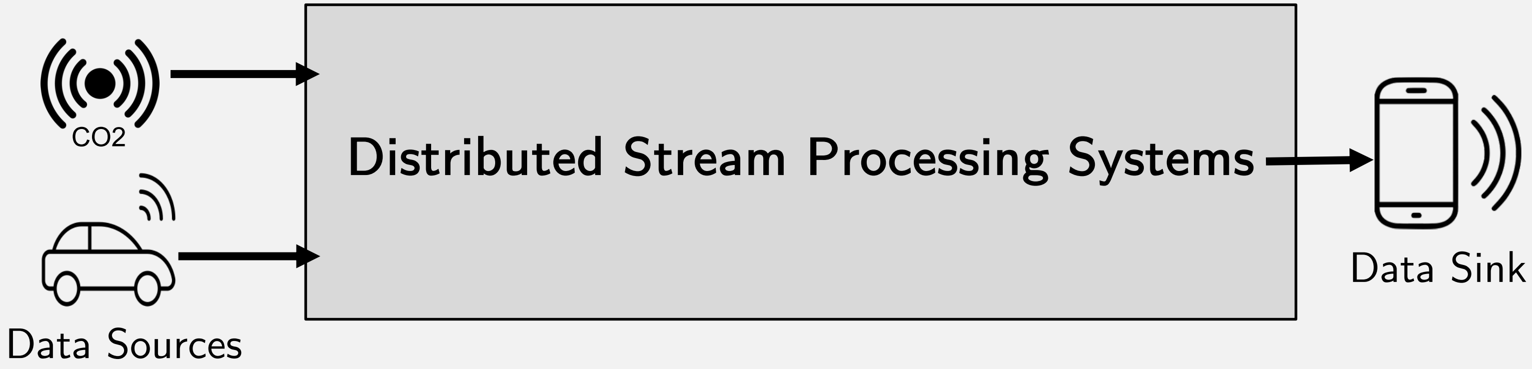
Stream Processing in IoT-Scenarios

Query: Analyze air pollution levels in cities and provide timely alerts to residents



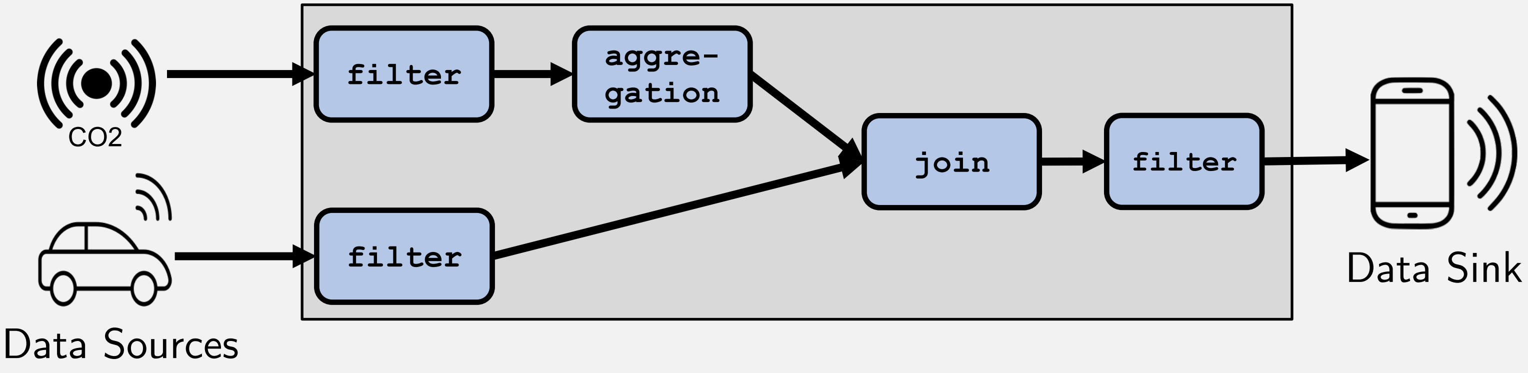
Stream Processing in IoT-Scenarios

Query: Analyze air pollution levels in cities and provide timely alerts to residents



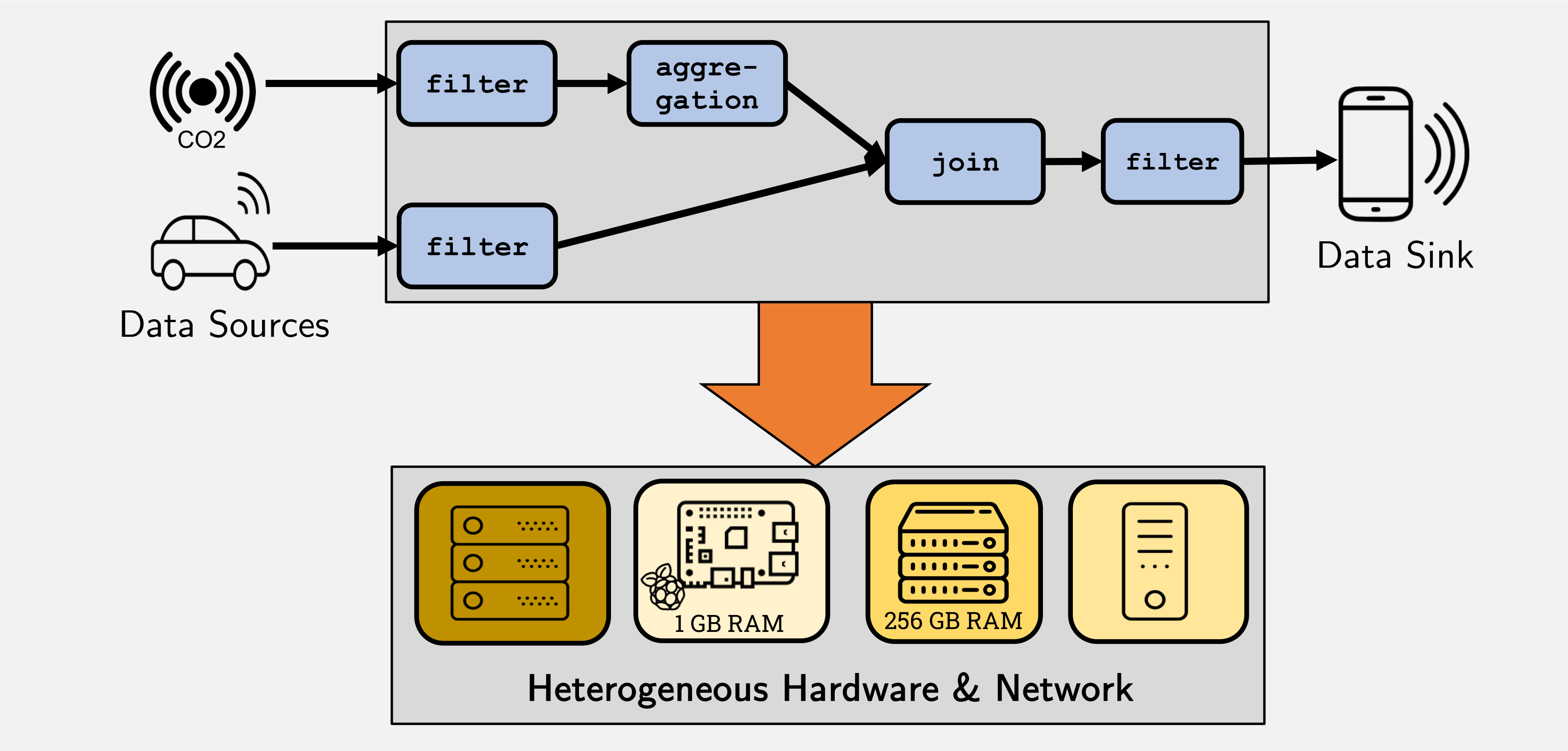
Stream Processing in IoT-Scenarios

Query: Analyze air pollution levels in cities and provide timely alerts to residents



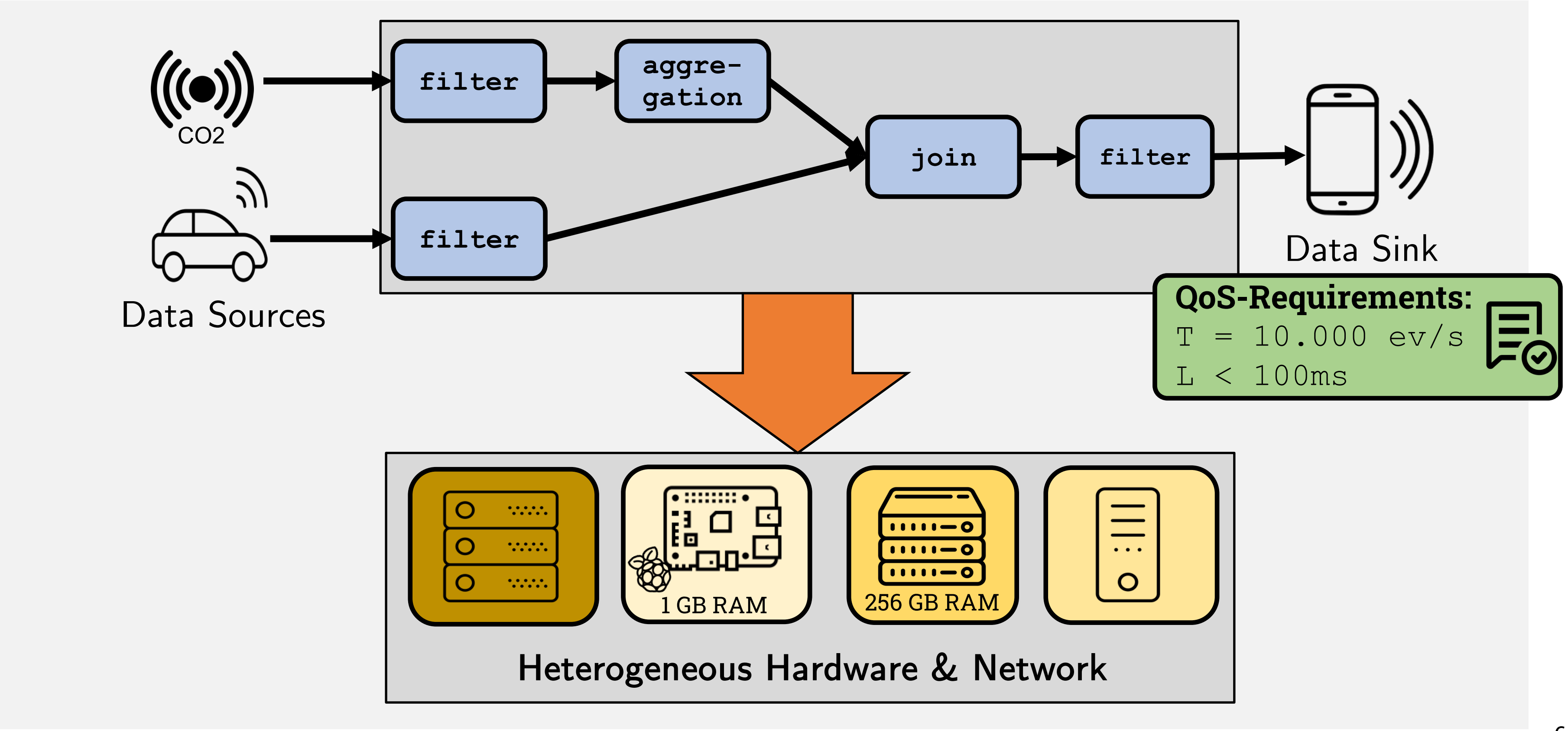
Stream Processing in IoT-Scenarios

Query: Analyze air pollution levels in cities and provide timely alerts to residents



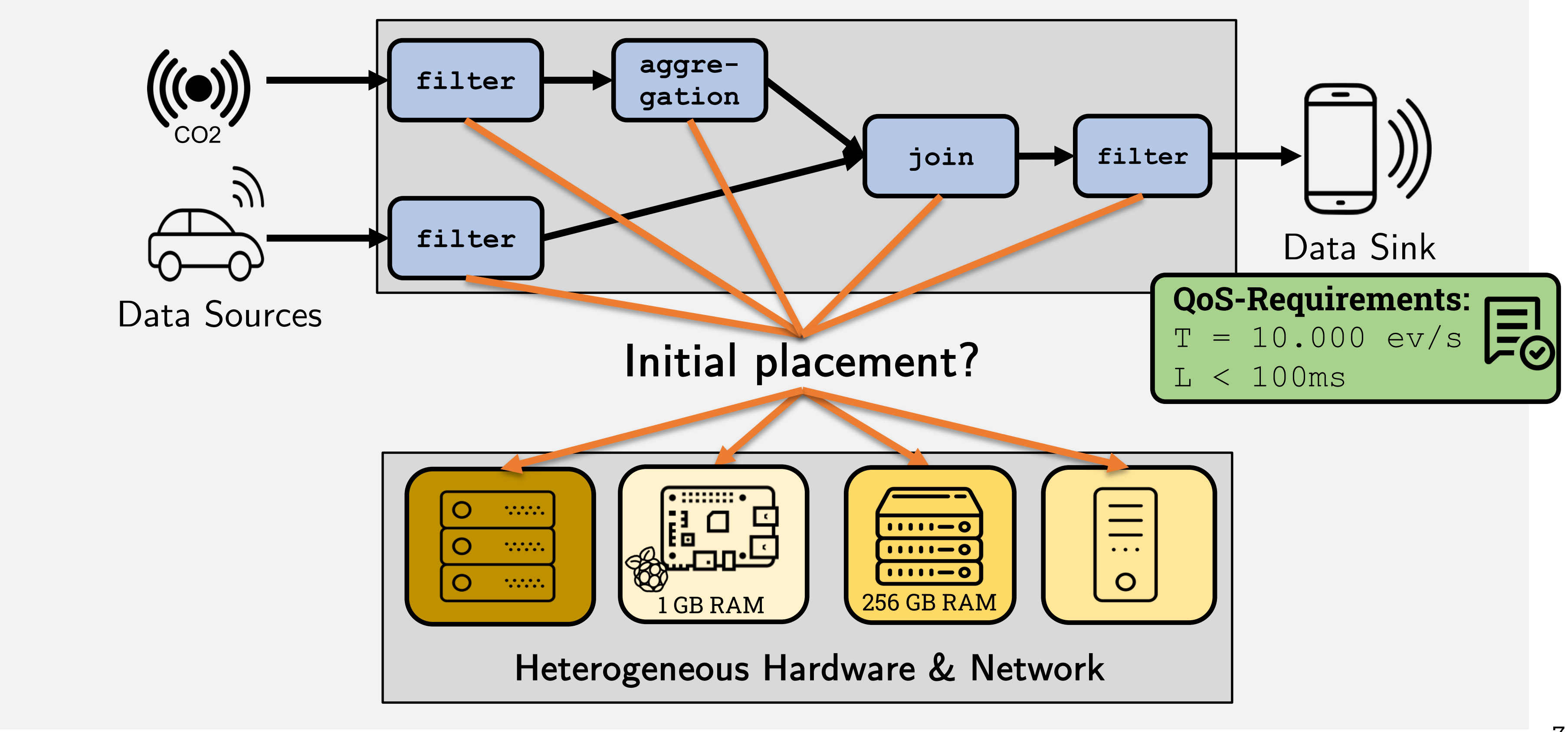
Stream Processing in IoT-Scenarios

Query: Analyze air pollution levels in cities and provide timely alerts to residents

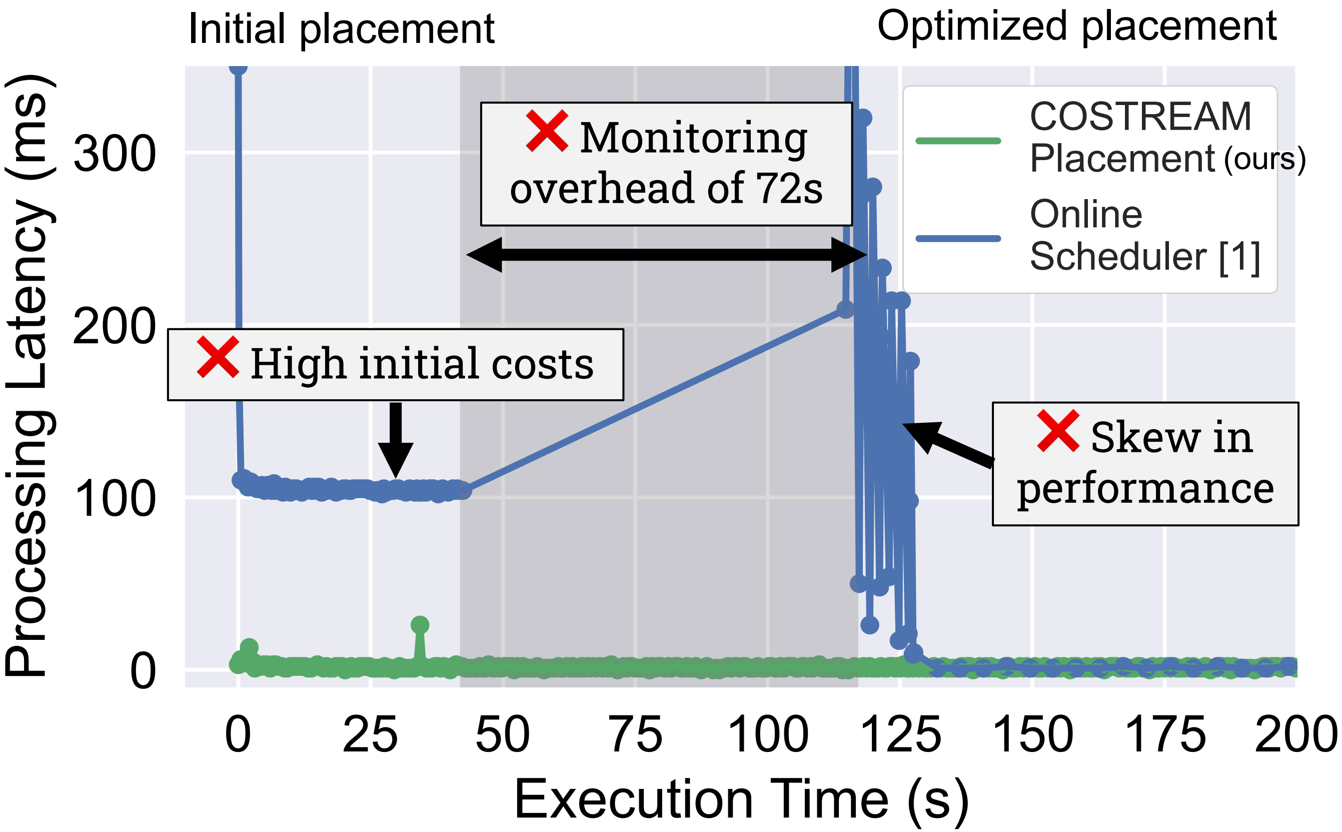


Stream Processing in IoT-Scenarios

Query: Analyze air pollution levels in cities and provide timely alerts to residents



Why the Initial Placement Matters



[1]: Aniello, *et al.* "Adaptive online scheduling in storm." *DEBS* '13.

Issues of Related Work

📈 **Online placement** [1]
❌ Long monitoring period
❌ Rescheduling downtime
❌ High initial cost

⚙️ **Heuristic-based placement** [2]
❌ Assume hardware homogeneity
❌ Inaccurate → sub-optimal placements

🤖 **Learned placement** [3]
❌ No generalization to unseen hardware, data & queries
⚠️ Especially required in IoT

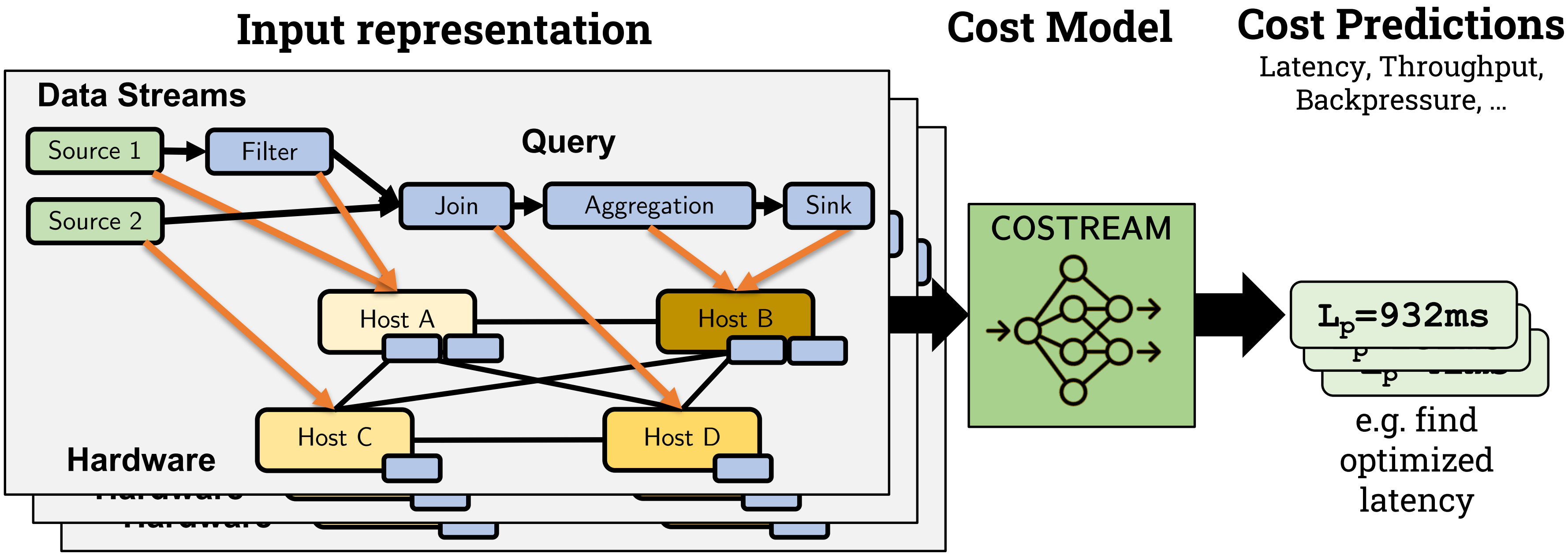
1. Enable **initial** placement
2. Predict **placement costs** for heterogeneous hardware accurately
3. Provide **Generalizabilty**

Research gaps

Goal: Finding an initial optimized placement that is generalizable

[1]: Aniello, *et al.* "Adaptive online scheduling in storm." *DEBS* `13.
[2]: Imai, *et al.* "Maximum sustainable throughput prediction for data stream processing over public clouds." CCGRID *2017*
[3]: Sun *et al.* "An end-to-end learning-based cost estimator." VLDB 2019

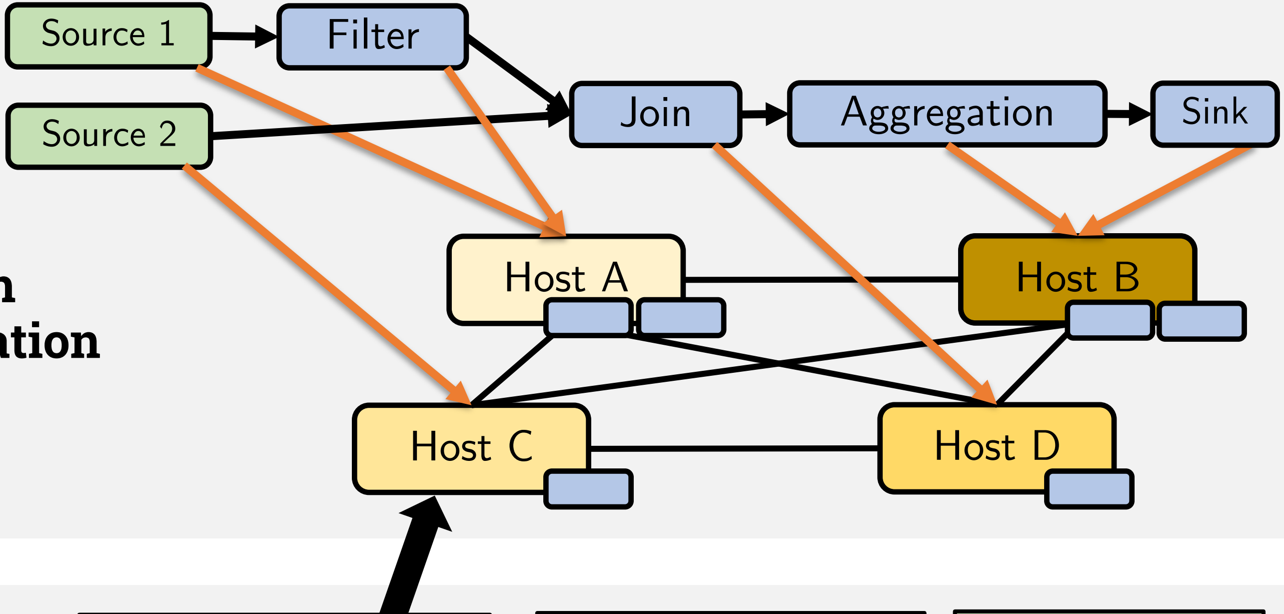
COSTREAM: A Novel Learned Cost Model



- COSTREAM enables **cost-based optimization** for DSPS.
- There is no **offline** cost model for stream processing yet.
- This work: **Placement optimization**

Transferable Input Representation

Novel joint graph representation

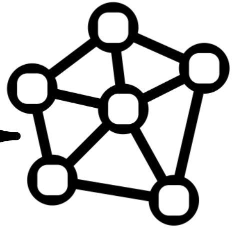


Transferable features

Hardware-related
CPU: 4 cores
RAM: 1024MB
Bandwidth: 20Mb/s
Latency: 5ms

Operator-related
Window length
Window Type
....

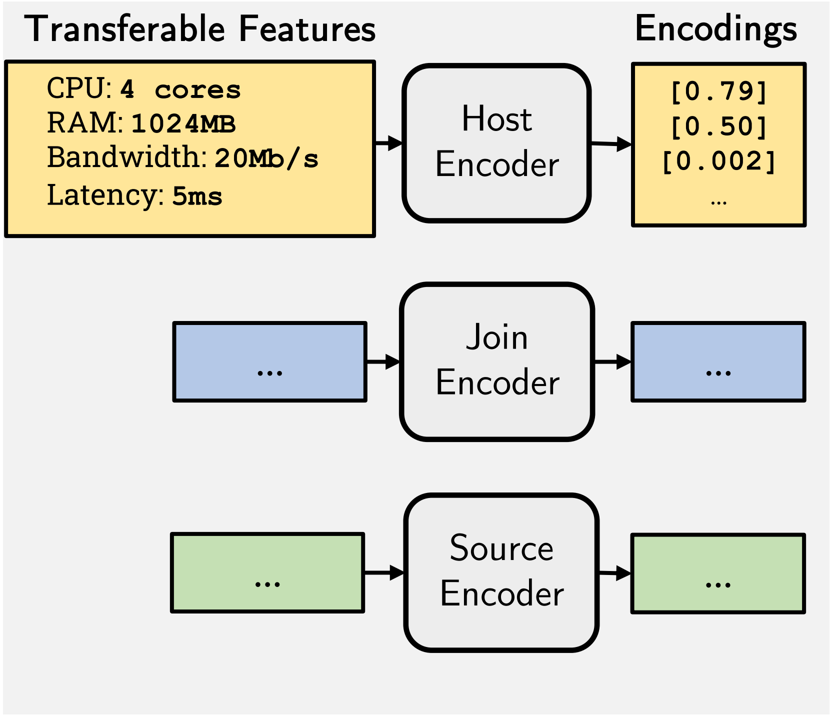
Data-related
Event rate
Selectivity
...



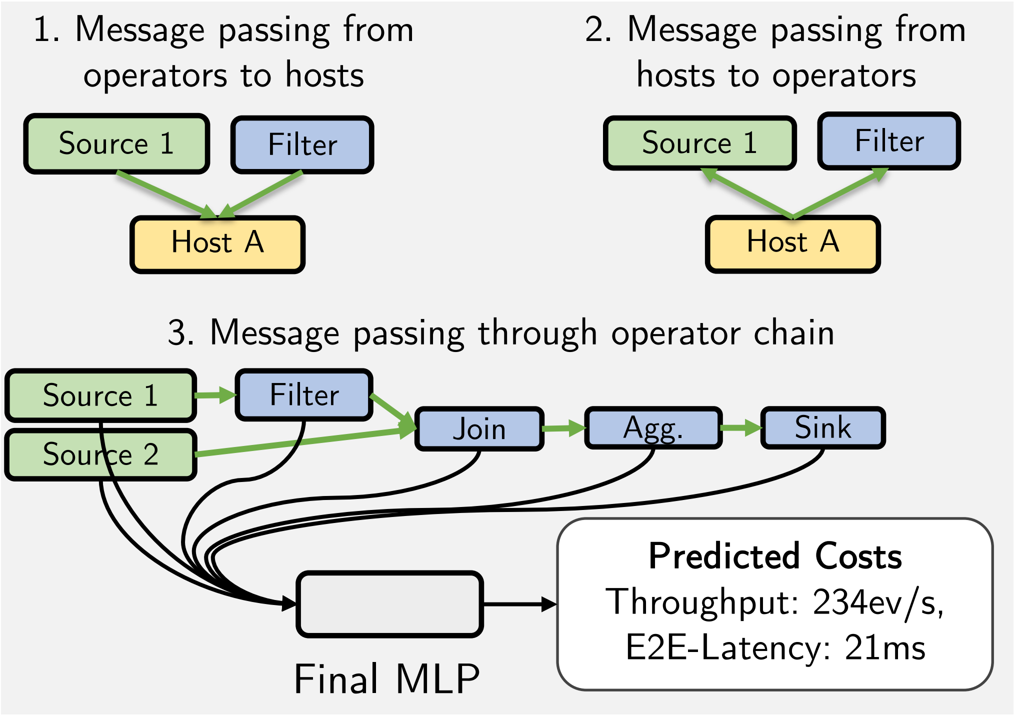
GNN

Learning Placement Costs with GNN

Neural Encoding



Novel Neural Message Passing



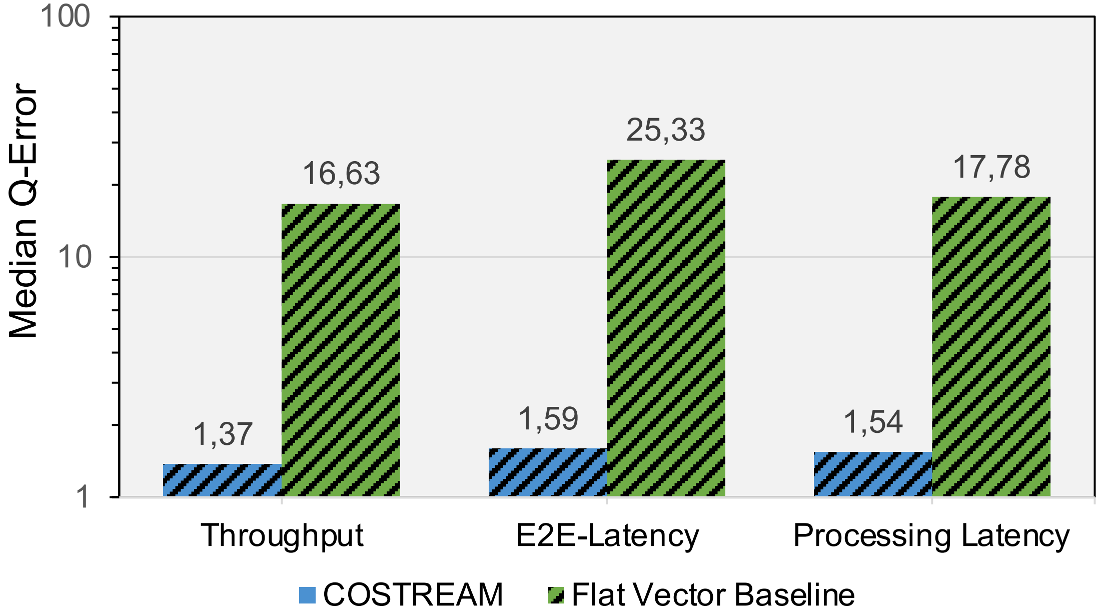
1. How good are cost predictions from COSTREAM?

- **Method:**
  Test predictions for **unseen** hardware that differs from initial training range..

- **Example:**
  Training - RAM:
  2, 4, 8 ,…
  Evaluation - RAM:
  3, 6, 12, …

- **Metric**:
  Deviation of real and predicted costs with median Q-Error:

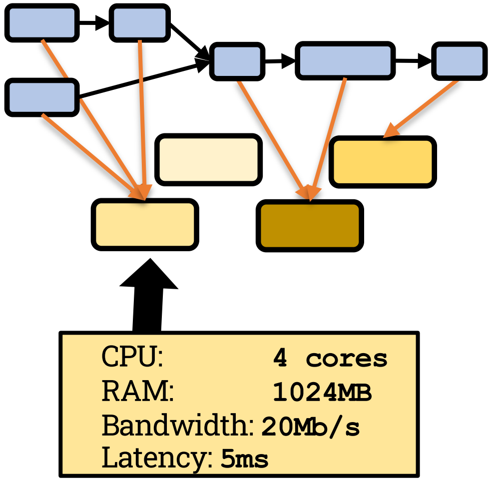
$$Q(x, x') = \max\left(\frac{x}{x'}, \frac{x'}{x}\right)$$

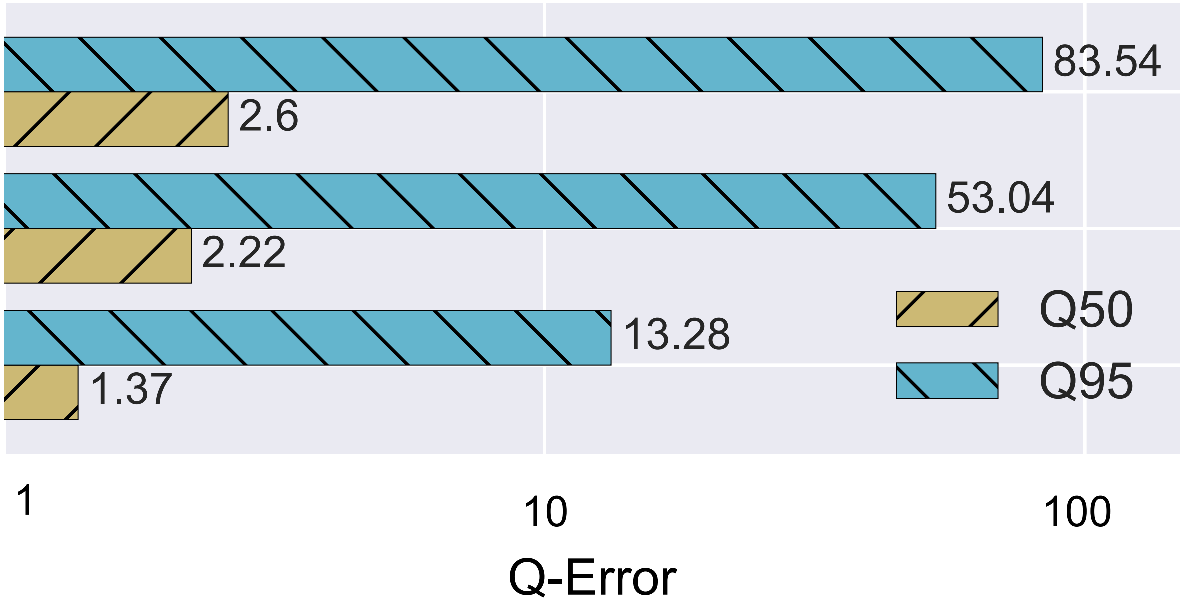


✅ Accurate predictions for unseen hardware

More Experiments : Unseen query types and data streams → in the paper

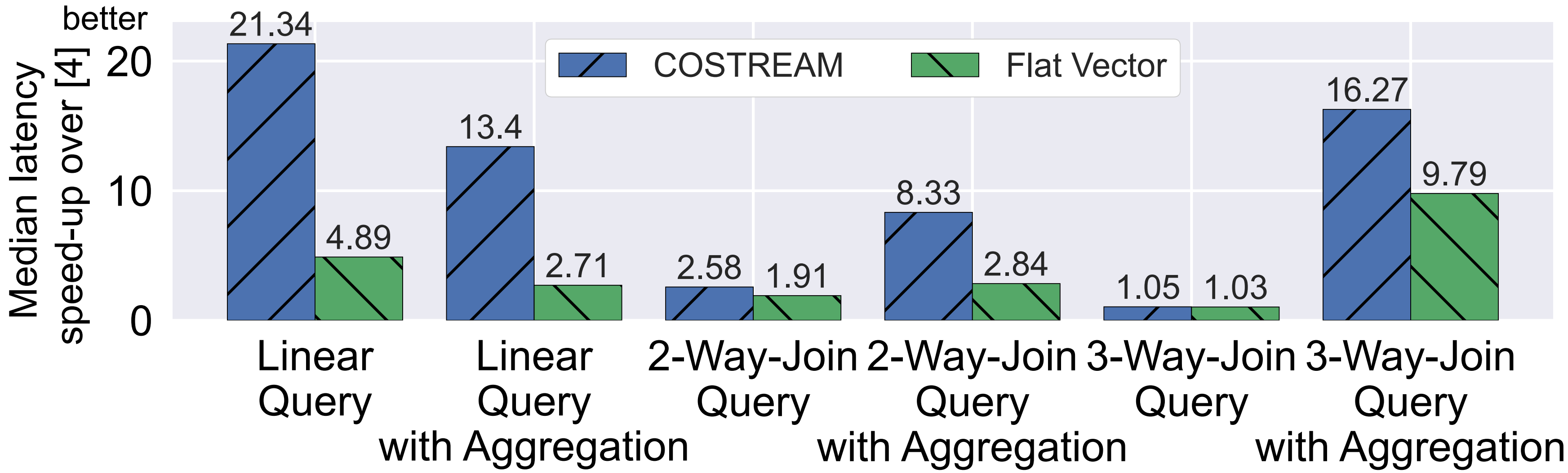
2. How much COSTREAM benefits from modeling heterogeneous hardware?





✅ Precise hardware modeling is highly beneficial

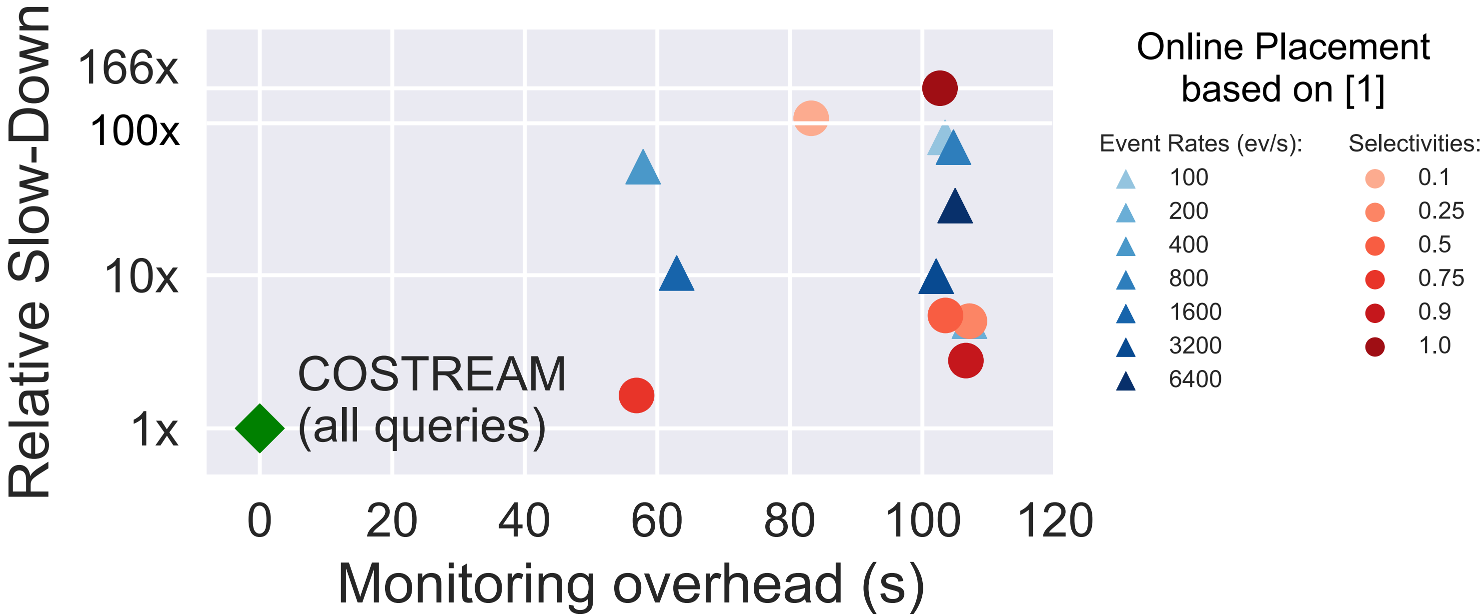
3. How good are the initial placements provided by COSTREAM?



✅ COSTREAM returns placements with high speed-ups across query types

[4]: Chaudhary, *et al.* "Governor: Operator Placement for a Unified Fog-Cloud Enrionment." *EDBT* '20.

4. What are benefits of cost-based placement optimization?



✅ High initial speed-ups of up to 166x

✅ Avoiding monitoring overhead of up to 120s

[1]: Aniello, *et al.* "Adaptive online scheduling in storm." *DEBS* '13.

Summary and Outlook

COSTREAM:

- … is a **novel learned model** for DSPS that predicts execution costs of the initial placement
- … shows advantages over **monitoring approaches**
- … is designed for **heterogeneous** hardware resources
- … generalizes to **unseen** queries, data streams and hardware
- … paves the way for cost based DSPS optimization

Next Steps:

- Bring cost-based optimization to other **DSPS tasks** like operator reordering
- Investigate generalizability **across DSPS**

Questions?

Paper **Code** **Data**

COSTREAM: Learned Cost Models for Operator Placement in Edge-Cloud Environments

Roman Heinrich, DHBW Mannheim
Carsten Binnig, TU Darmstadt & DFKI
Harald Kornmayer, DHBW Mannheim
Manisha Luthra, TU Darmstadt & DFKI

***Abstract*—In this work, we present COSTREAM, a novel learned cost model for Distributed Stream Processing Systems that provides accurate predictions of the execution costs of a streaming query in an edge-cloud environment. The cost model can be used to find an initial placement of operators across heterogeneous hardware, which is particularly important in these environments. In our evaluation, we demonstrate that COSTREAM can produce highly accurate cost estimates for the initial operator placement and even generalize to *unseen* placements, queries, and hardware. When using COSTREAM to optimize the placements of streaming operators, a median speed-up of around 21× can be achieved compared to baselines.**

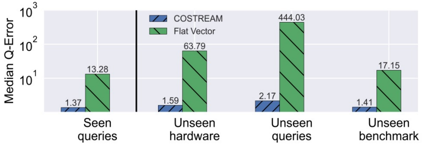
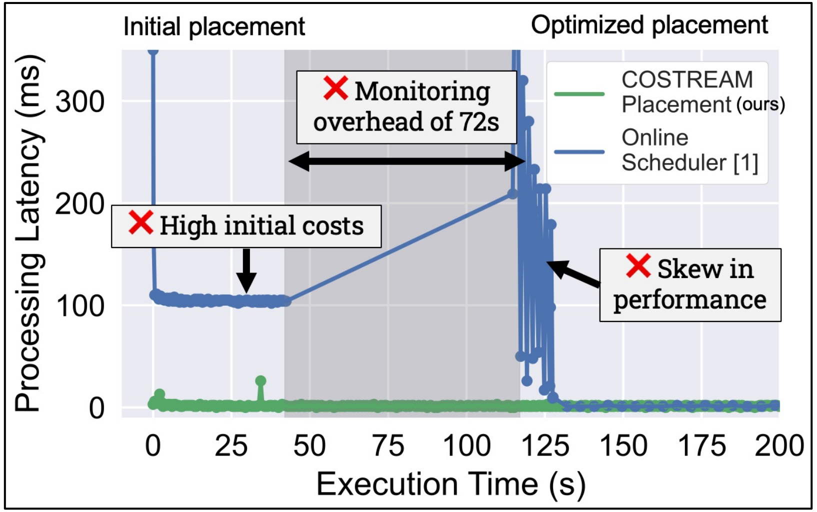


Fig. 1: Estimation errors when predicting E2E-latency for queries that are similar to the training data (left) or entirely unseen in terms of underlying hardware and other query properties (right). COSTREAM can precisely predict query execution costs compared to an existing cost model baseline (Flat Vector).



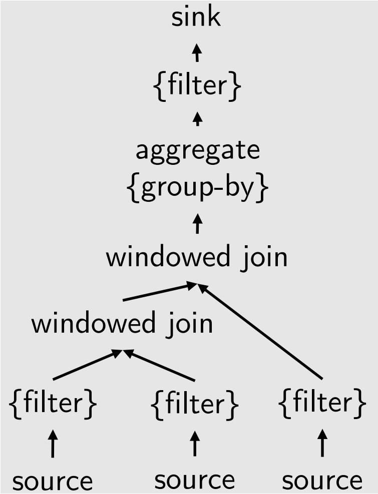
Back-Up Slides

Training Benchmark

Benchmark with 43.281 queries

- Various query templates
- Various data streams
- Various hardware resources
- Various placements based on heuristics [3]

3-way-join



[3] Chaudhary *et al.*, "Governor: Operator placement for a unified fog-cloud environment," *EDBT 2020*

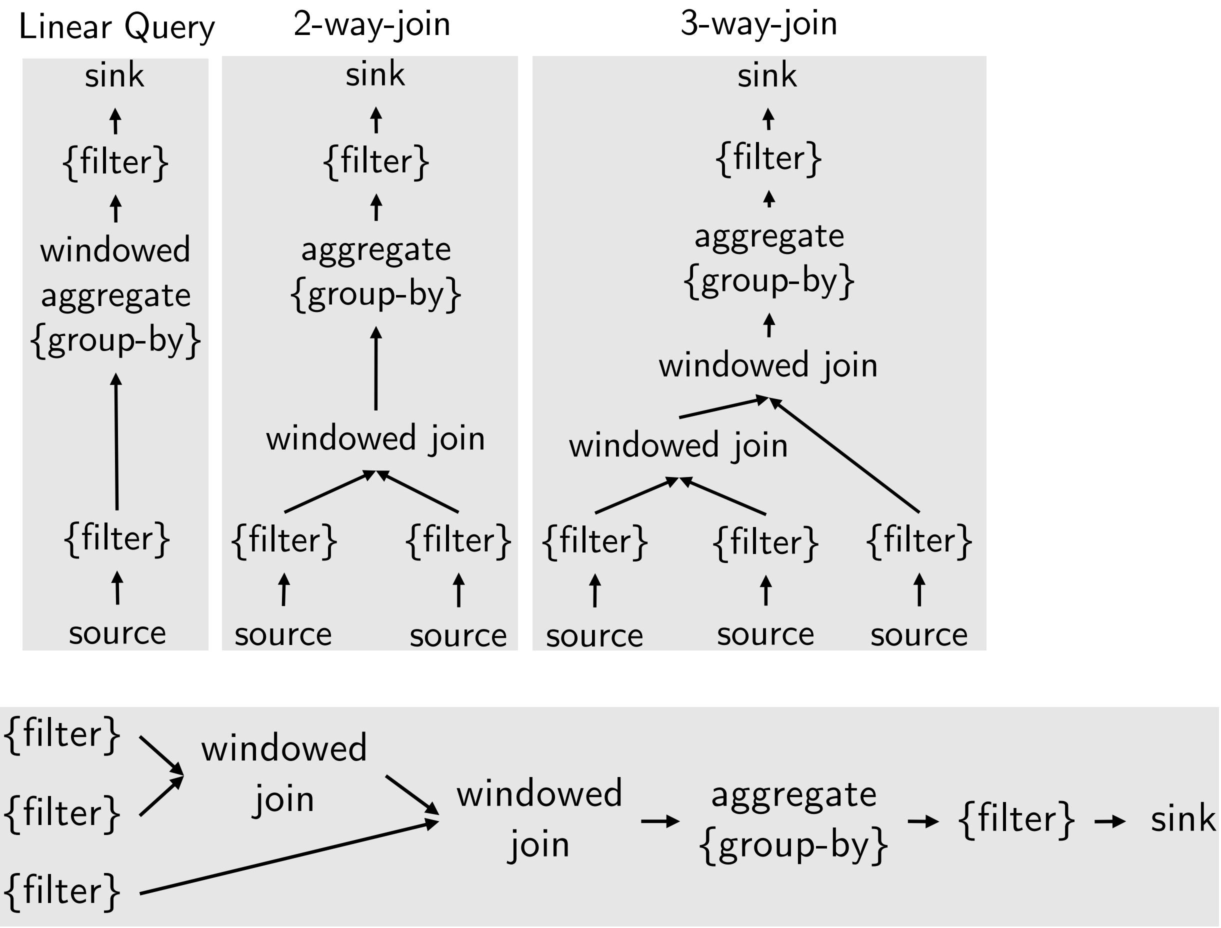
Transferable Features

Node	Category	Feature	Description
all	data	tuple width in	Averaged incoming tuple width
	data	tuple width out	Outgoing tuple width
source	data	input event rate	Event rate emitted by the source
	data	tuple data type	Data type for each value in tuple
filter	operator	filter function	Comparison function
	operator	literal data type	Data type of comparison literal
	data	selectivity	see Definition 6
join	operator	join-key data type	Data type of the join key
	data	selectivity	see Definition 7
agg.	operator	agg. function	Aggregation function
	operator	group-by data type	Data type of group-by attribute
	operator	agg. data type	Data type of each value to aggregate
	data	selectivity	see Definition 8
window	operator	window type	Shifting strategy (sliding/tumbling)
	operator	window policy	Counting mode (count/time-based)
	operator	window size	Size of the window
	operator	slide size	Size of the sliding interval
hardware	hardware	cpu	Available CPU resources in %
	hardware	ram	Available RAM resources in MB
	hardware	network-latency	Outgoing latency of the host in ms
	hardware	network-bandwidth	Outgoing bandwidth of the host in Mbit/s

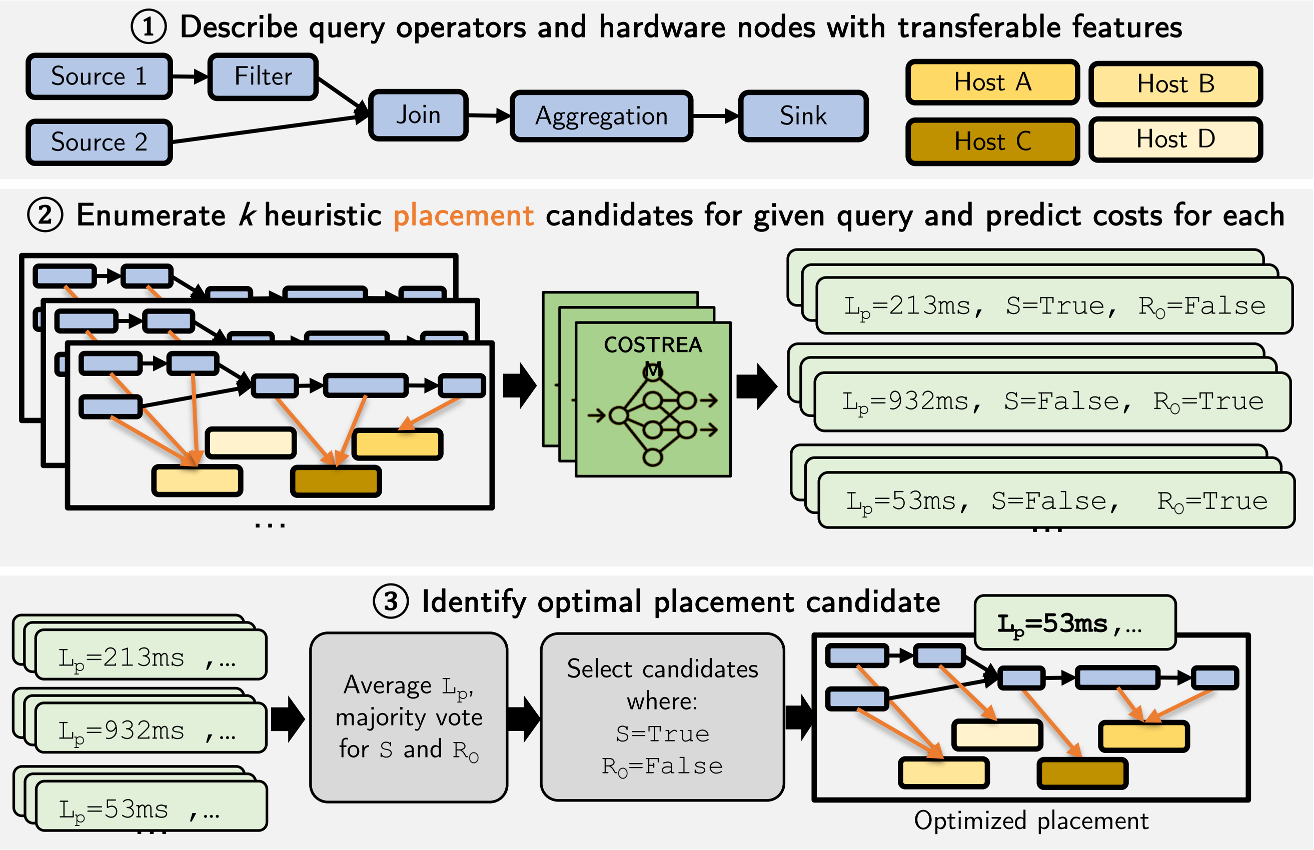
Feature Range of Benchmark

Feature	Training data range
cpu	[50, 100, 200, 300 400, 500, 600, 700, 800] % of a core
ram	[1000, 2000, 4000, 8000, 16000, 24000, 32000] MB
network bandwidth	[25, 50, 100, 200, 400, 800, 1600, 3200, 6400, 10000] MBits
network latency	[1, 2, 5, 10, 20, 40, 80, 160] ms
input event rate (linear)	[100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600] ev/s
input event rate (two-way)	[50, 100, 250, 500, 750, 1000, 1250, 1500, 1750, 2000] ev/s
input event rate (three-way)	[20, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000] ev/s
tuple data type	[3...10] × [int, string, double]
filter function	<,>,<=, >=, !=, startswith, endswith
literal data type	int, string, double
window type	sliding, tumbling
window policy	count-based, time-based
window size (count)	[5, 10, 20, 40, 80, 160, 320, 640] tuples
window size (time)	[0.25, 0.5, 1, 2, 4, 8, 16] sec
slide size	[0.3 ... 0.7] × window length
join-key data type	int, string, double
agg. function	min, max, mean, avg
group-by data type	int, string, double, none

Query Examples

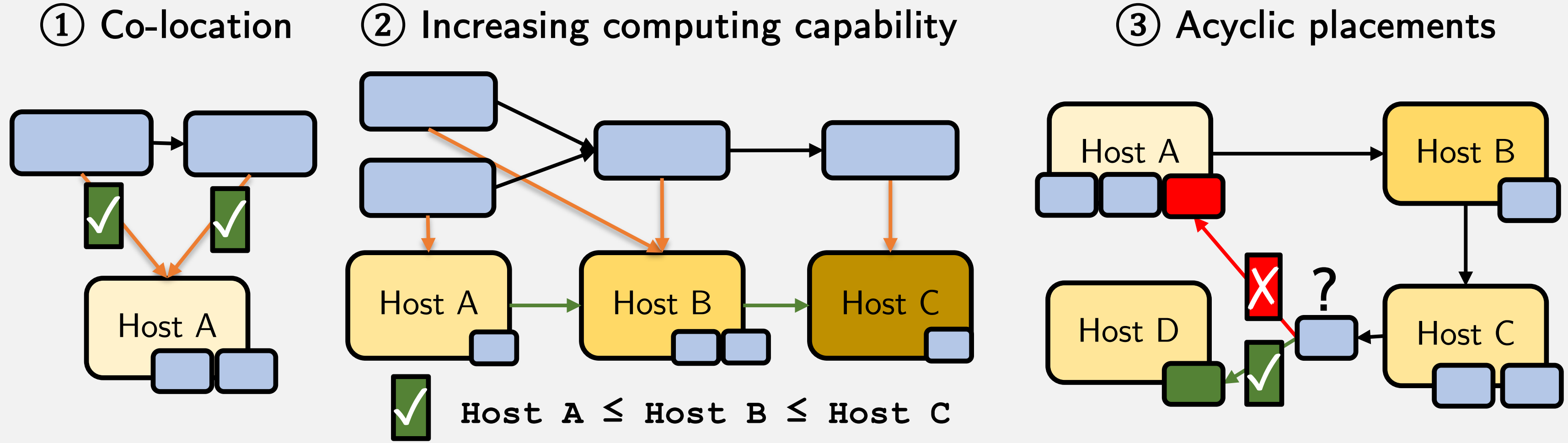


Optimization Procedure



Placement heuristics

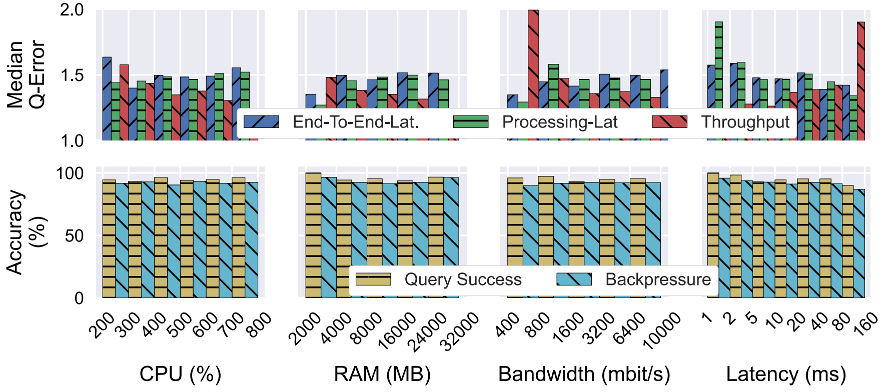
Placement enumeration: Based on published heuristics [3]



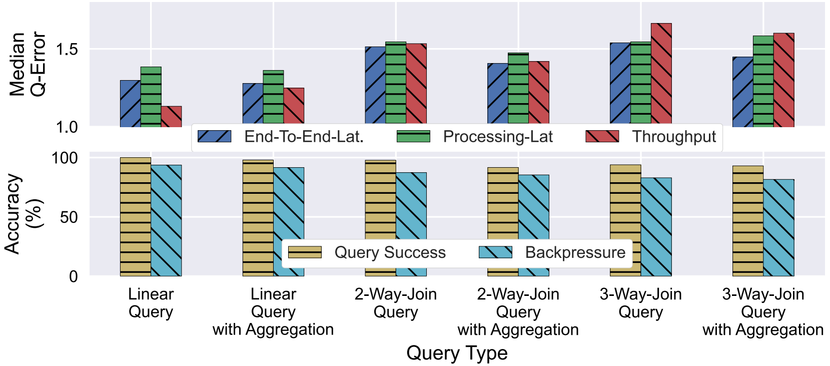
[3] Chaudhary *et al.*, "Governor: Operator placement for a unified fog-cloud environment," *EDBT 2020*

General Prediction Accuracy

Hardware properties

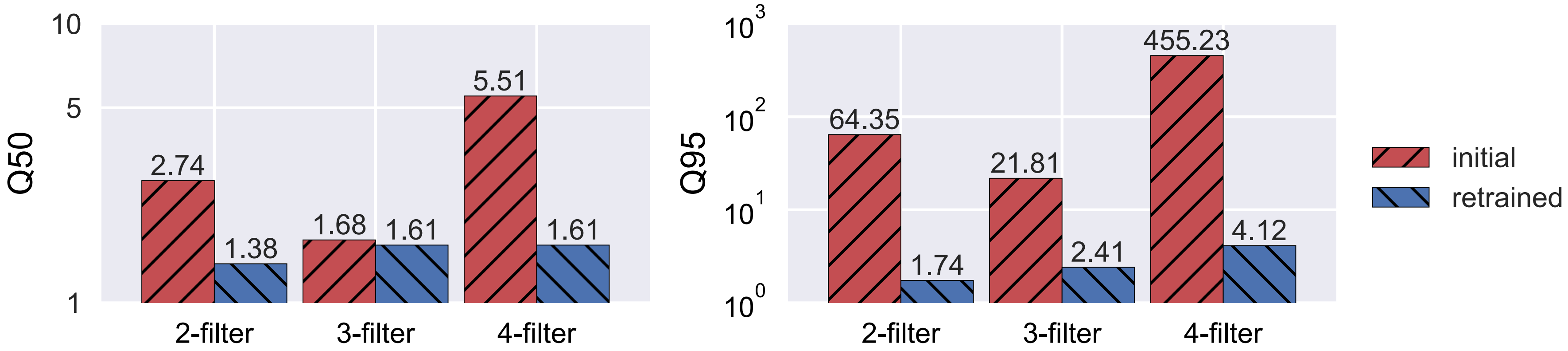


Query Type



	COSTREAM		FLAT VECTOR	
Metric	**Q50**	Q95	**Q50**	Q95
Throughput	**1.33**	5.60	9.92	590.34
E2E-latency	**1.37**	13.28	24.96	827.59
Processing latency	**1.46**	13.90	22.87	458.14
Backpressure	**87.89%**		68.70%	
Query success	**94.96%**		76.85%	

Few-Shot Learning Improves Results



Re-Training COSTREAM with a few target queries for filter chains

Extrapolation Towards Hardware

Ⓐ **Extrapolation towards stronger resources**

	RAM (GB)		**CPU (% of a core)**		**Bandwidth (Mbit/s)**		**Latency (ms)**	
Training Range	1, 2, 4, 8, 16		50, 100, 200, 300, 400, 500, 600		25, 50, 100, 200, 300, 800, 1.6k, 3.2k		5, 10, 20, 40, 80, 160	
Evaluation Range	24, 32		700, 800		64k, 10k		1, 2	
Metric	**Q50**	Q95	**Q50**	Q95	**Q50**	Q95	**Q50**	Q95
Throughput	**1.66**	5.88	**1.72**	9.40	**1.48**	6.55	**1.52**	5.60
E2E-Latency	**1.85**	29.08	**1.67**	9.43	**1.75**	17.18	**3.55**	30.90
Processing Latency	**1.88**	11.32	**1.75**	6.81	**1.63**	13.89	**3.83**	19.43
Backpressure	**85.37%**		**86.59%**		**86.59%**		**88.89%**	
Query Success	**77.00%**		**93.14%**		**87.25%**		**92.93%**	

Ⓑ **Extrapolation towards weaker resources**

	RAM (GB)		**CPU (% of a core)**		**Bandwidth (Mbit/s)**		**Latency (ms)**	
Training Range	4, 8, 16, 24, 32		200, 300, 400, 500, 600, 700, 800		100, 200, 300, 800, 1.6k, 3.2k, 6.4k, 10k		1,2,5,10 20, 40	
Evaluation Range	1, 2		50, 100		25, 50		80, 160	
Metric	**Q50**	Q95	**Q50**	Q95	**Q50**	Q95	**Q50**	Q95
Throughput	**1.79**	7.60	**1.61**	13.16	**1.42**	5.30	**3.25**	33.65
E2E-Latency	**1.72**	13.69	**2.75**	111.53	**1.46**	5.30	**2.10**	54.13
Processing Latency	**1.49**	13.27	**2.96**	77.56	**1.68**	12.94	**6.09**	406.83
Backpressure	**91.03%**		**75.00%**		**91.92%**		**67.82%**	
Query Success	**78.79%**		**86.67%**		**92.59%**		**74.51%**	

Interpolation Results

Ⓐ	RAM (GB)	CPU (% of a core)	Bandwidth (Mbit/s)	Latency (ms)
Training Range	1, 2, 4, 8, 16, 24, 32	50, 100, 200, 300, 400, 500, 600, 700, 800	25, 50, 100, 200, 300, 800, 1600, 3200, 4800, 800	1, 2, 5, 10, 20, 40, 80, 160
Evaluation Range	1.5, 3, 6, 12, 20, 28	75, 150, 250, 350, 450, 550, 650, 750	35, 75, 150, 250, 550, 1200, 1900, 4800, 8000	3, 7, 15, 30, 60, 120

Ⓑ	COSTREAM		Flat Vector	
Metric	**Q50**	Q95	**Q50**	Q95
Throughput	**1.37**	8.28	**15.63**	282.50
E2E-Latency	**1.59**	25.33	**63.79**	869.85
Processing Latency	**1.54**	17.78	**27.85**	282.50
Backpressure	**88.04%**		72.83%	
Query Success	**87.13%**		68.32%	

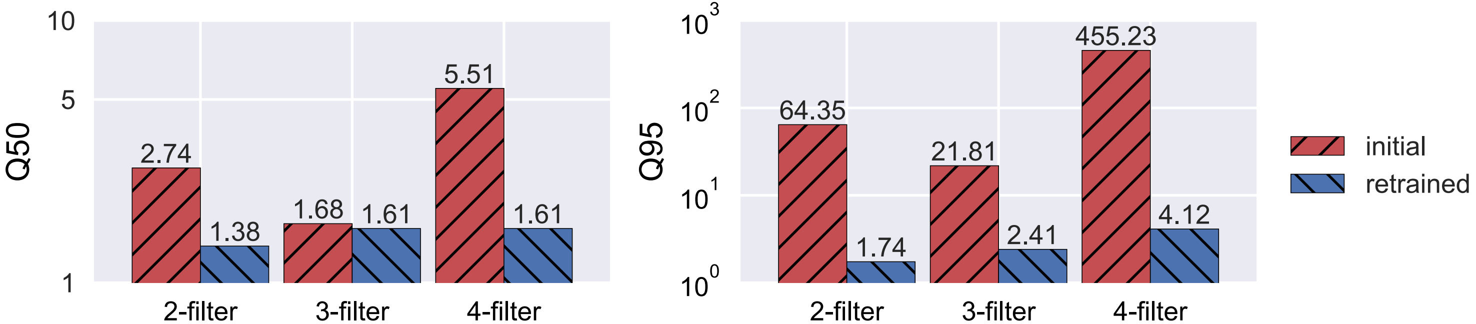
Extrapolation Towards Unseen Queries

Ⓐ **[Exp 5] Unseen query pattern**

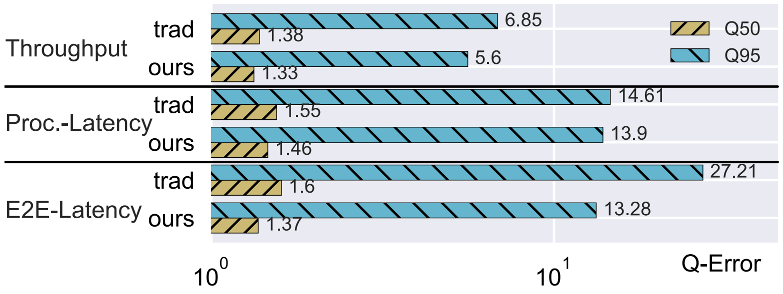
	2-Fiter Chain				**3-Filter-Chain**				**4-Filter-Chain**			
	COSTREAM		FLAT VECTOR		COSTREAM		FLAT VECTOR		COSTREAM		FLAT VECTOR	
Metric	Q50	Q95	Q50	Q95	Q50	Q95	Q50	Q95	Q50	Q95	Q50	Q95
Throughput	**2.74**	64.35	5.52	244.38	**2.87**	75.29	18.82	1078.26	**5.51**	445.87	82.71	3672.13
E2E-Latency	**1.68**	21.81	259.98	2302.38	**2.15**	11.81	536.38	1855.05	**2.68**	23.99	538.10	1877.68
Proc-Latency	**1.69**	48.26	48.93	341.70	**1.64**	5.41	63.62	266.80	**1.61**	5.38	55.27	270.36
Backpressure	**88%**		68%		**85%**		79%		**82%**		79%	
Query success	**100%**		4%		**100%**		6%		**100%**		6%	

Ⓑ **[Exp 6] Unseen benchmarks**

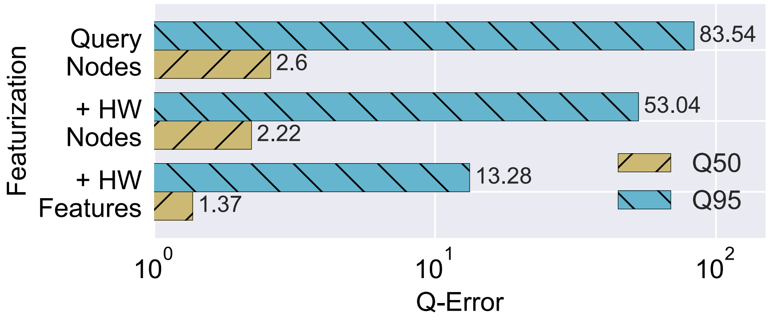
Advertisement				**Spike Detection**				**Smart Grid (global)**				**Smart Grid (local)**			
COSTREAM		FLAT VECTOR		COSTREAM		FLAT VECTOR		COSTREAM		FLAT VECTOR		COSTREAM		FLAT VECTOR	
Q50	Q95	Q50	Q95	Q50	Q05	Q50	Q95	Q50	Q95	Q50	Q95	Q50	Q95	Q50	Q95
1.98	11.01	3.12	46.11	**3.67**	66.48	274.04	891.99	**1.44**	5.98	104.79	106.06	**1.43**	10.51	104.79	106.12
2.02	15.08	**1.32**	40.59	**1.41**	17.55	2.28	1017.96	**2.01**	50.17	118.77	639.79	**1.67**	31.00	143.22	669.20
2.27	15.01	3.62	41.37	**1.63**	12.92	5.32	339.82	**1.48**	12.70	35.48	161.60	**1.54**	7.96	37.57	174.38
85%		80%		**78%**		55%		**81%**		29%		**86%**		23%	
100%		100%		100%		0%		**100%**		100%		**100%**		100%	



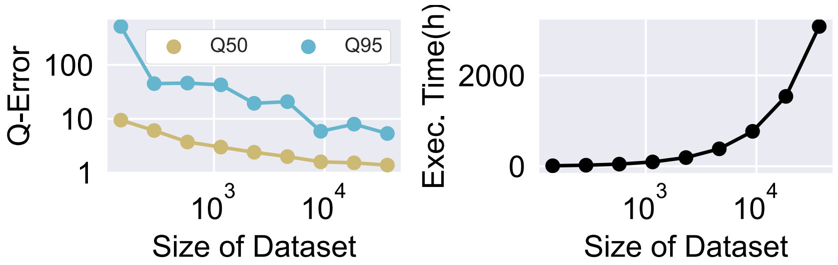
Ablation Studies



Message passing scheme

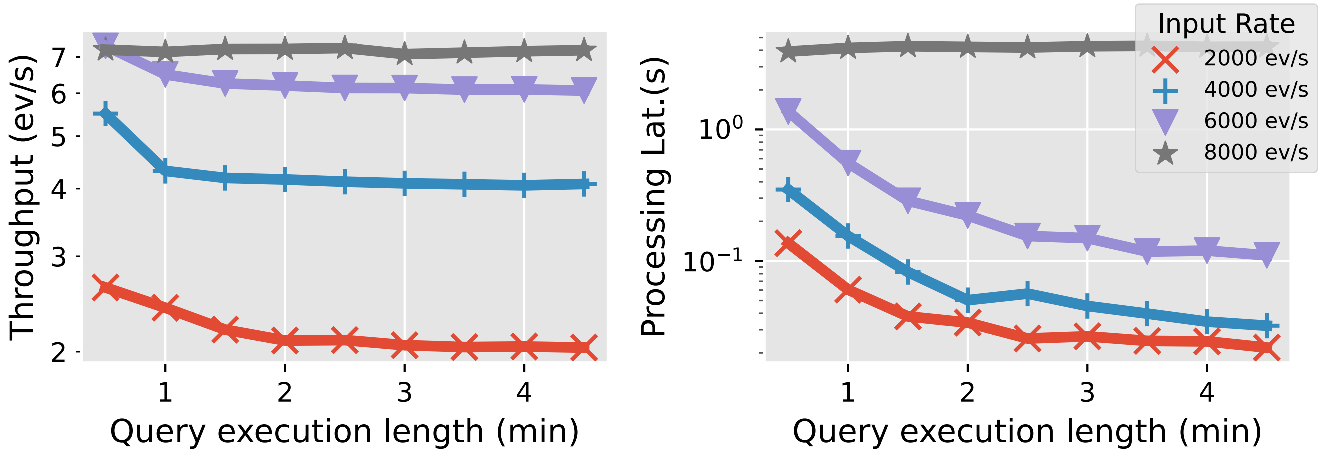


Featurization



Learning Curve

Query execution length



Query execution costs over load