Jiesong Liu

Computer Science Department, North Carolina State University

February 13, 2025

Jiesong Liu

- Ph.D student in Computer Science at North Carolina State University, 2023 - now
- B.E. at Renmin University of China, 2019 - 2023.
- Research Assistant to Prof. Feng Zhang, Renmin University of China
  Feb. 2021 - Sept. 2021, topic: Parallel Computing, Database Systems
- Research Assistant to Prof. Xipeng Shen, North Carolina State University
  Sept. 2021 - now, topic: AI Optimization

LLM Inference Optimization

- Googles infrastructure optimizations show that a 1% speedup in inference can save millions.
- Speculative Decoding breaks the autoregressive nature of LLMs and allows execution in parallel, by first running inference with a smaller LLM to generate the next γ tokens, and verifying those tokens using the large LLM.
- Choosing the best speculation window length, γ, and the best draft model to use is crucial for unlocking the potential of speculative decoding.

Design

- Introduce *on-the-fly adaptive speculation*, a drop-in solution that adapts speculative decoding at runtime without ahead-of-time training.

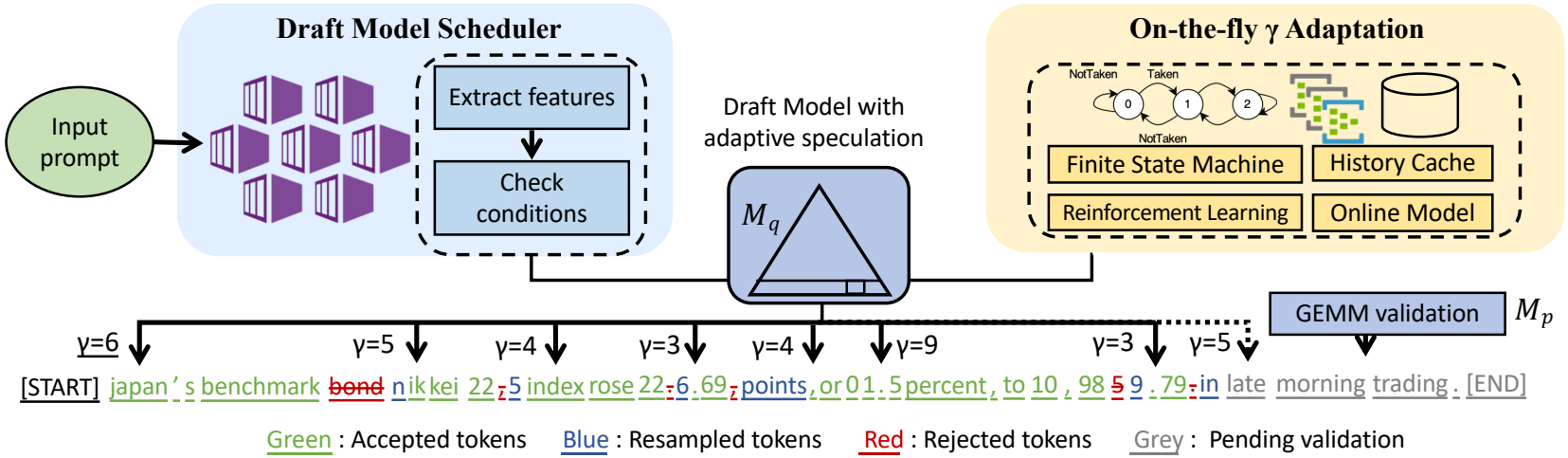


Figure 1: Our on-the-fly adaptive speculation framework. When a prompt arrives, our scheduler directs it to the draft model M_q. During speculation, our framework automatically adapts the right speculation window size γ. The speculation is then validated by the target model M_p.

Online window size optimization

Definition (formulating objective)

Let a_q represent the latency of generating one token by the draft model, and $b_p(\gamma)$ represent the latency of a verification step with window size γ. For $t = 1, 2, \cdots$, let $Acc(x_t|X_{<t})$ be the accuracy of the speculation of a single token given the current context $X_{<t} = \{x_1, \cdots, x_{t-1}\}$. The window size γ for the current speculation step can be determined by optimizing the objective

$$\mathcal{G} = \max_{\gamma} \frac{1 - Acc(x_t|X_{<t})^{\gamma+1}}{(1 - Acc(x_t|X_{<t}))(\gamma a_q + b_p(\gamma))}. \tag{1}$$

We estimate $Acc(x_t|X_{<t})$ as

$$\frac{\sum_j V(\gamma(j), X_{<t_j})}{\sum_j V(\gamma(j), X_{<t_j}) + \sum_j \mathbf{1}(V(\gamma(j), X_{<t_j}) \leq \gamma(j))} \tag{2}$$

Results

Table 1: Evaluation of adaptive window size selection. SPS denotes the throughput improvement our method achieves over the original speculative decoding. ARS denotes improvements over the default LLMs without speculative decoding. ("-" for not-runnable cases due to memory limit)

Model Pairing	Dataset	A100		V100		4090	
		SPS	ARS	SPS	ARS	SPS	ARS
LLaMA 70B/7B	finance-alpaca	6.43%	2.11×	-	-	-	-
LLaMA 70B/13B	finance-alpaca	4.89%	1.90×	-	-	-	-
BLOOM 7B/560M	finance-alpaca	4.28%	1.05×	7.69%	1.15×	3.70%	1.22×
BLOOM 7B/1B1	finance-alpaca	4.36%	1.04×	3.20%	1.15×	3.29%	1.17×
OPT 13B/125M	finance-alpaca	4.82%	2.32×	3.41%	3.4×	-	-
Dolly 12B/3B	finance-alpaca	9.11%	1.03×	-	-	-	-
LLaMA 70B/7B	humaneval	10.35%	2.41×	-	-	-	-
LLaMA 70B/13B	humaneval	8.53%	2.23×	-	-	-	-
BLOOM 7B/560M	humaneval	8.14%	1.04×	2.51%	1.09×	3.09%	1.25×
BLOOM 7B/1B1	humaneval	4.03%	1.1×	3.57%	1.16×	3.51%	1.3×
OPT 13B/125M	humaneval	11.40%	2.29×	2.15%	3.34×	-	-
Dolly 12B/3B	humaneval	15.20%	1.07×	-	-	-	-
LLaMA 70B/7B	gsm8k	7.13%	2.28×	-	-	-	-
LLaMA 70B/13B	gsm8k	9.66%	2.08×	-	-	-	-
BLOOM 7B/560M	gsm8k	15.03%	1.×	2.52%	1.01×	4.84%	1.18×
BLOOM 7B/1B1	gsm8k	10.70%	1.×	0.77%	1.02×	1.97%	1.19×
OPT 13B/125M	gsm8k	5.95%	2.24×	10.52%	3.36×	-	-
Dolly 12B/3B	gsm8k	16.92%	1.06×	-	-	-	-
LLaMA 70B/7B	xsum	2.94%	1.73×	-	-	-	-
LLaMA 70B/13B	xsum	0.14%	1.5×	-	-	-	-
BLOOM 7B/560M	xsum	77.50%	1.×	49.30%	1.×	54.63%	1.×
BLOOM 7B/1B1	xsum	70.91%	1.×	42.94%	1.×	54.17%	1.×
OPT 13B/125M	xsum	10.64%	1.02×	7.91%	2.43×	-	-

Results

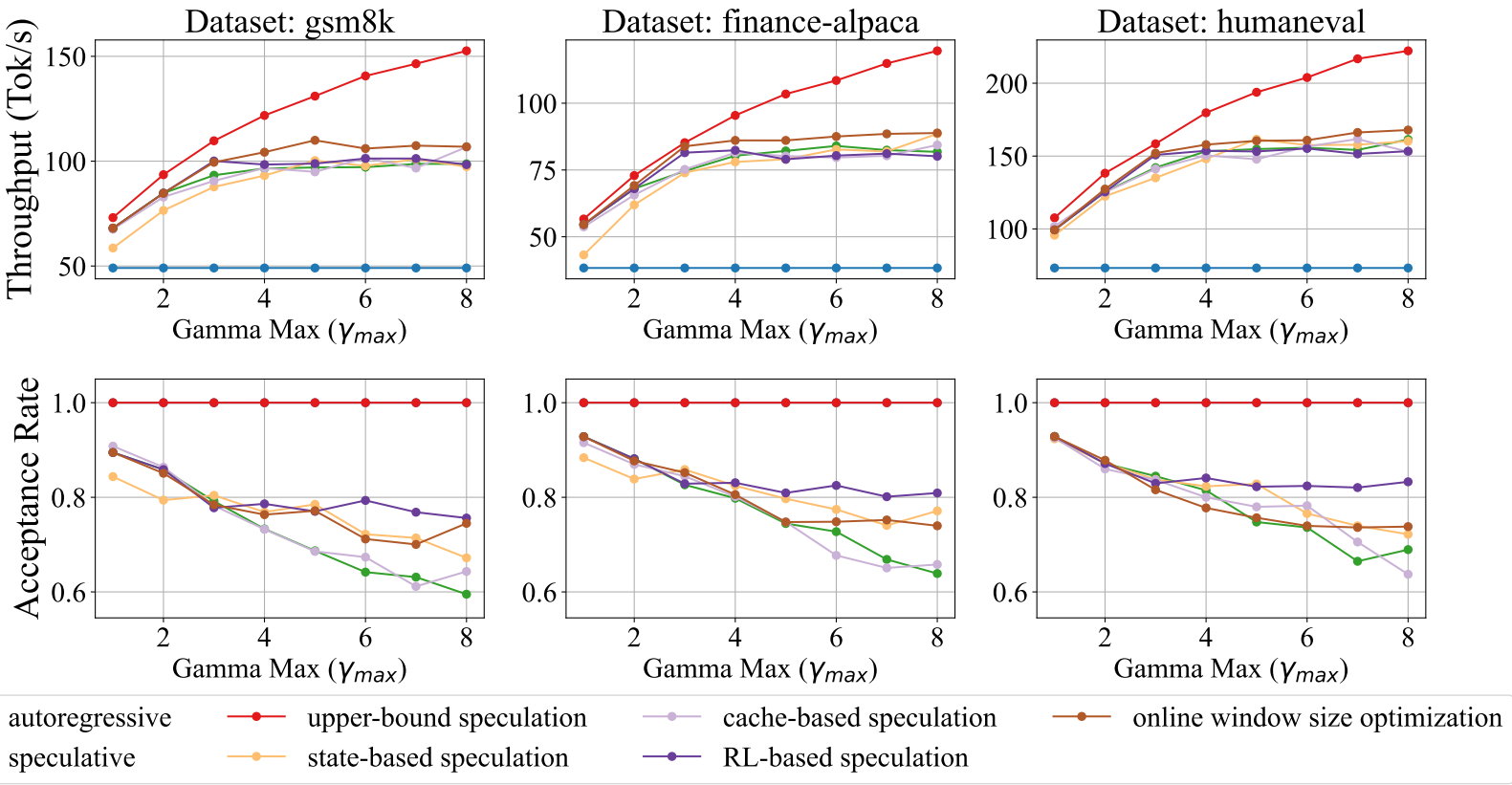


Figure 2: Detailed experimental results of different adaptive methods.

Background

- Microcontrollers are small, low-cost or energy-efficient devices.
- Demands for DNNs on microcontrollers keep growing for reasons of energy-saving and privacy concerns.

Background

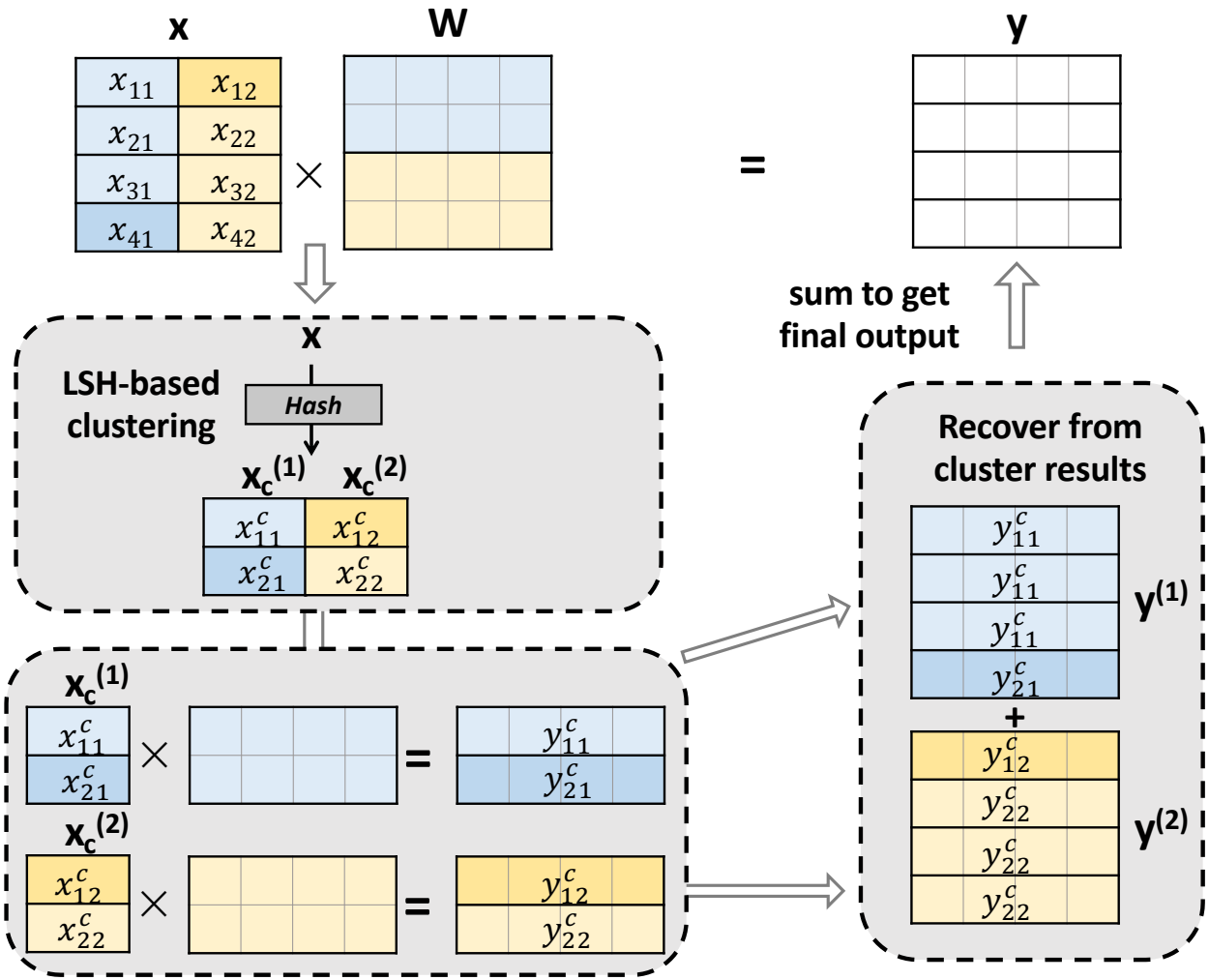


Figure 3: Transient redundancy elimination-based convolution (TREC) Method (for Inference).

Design

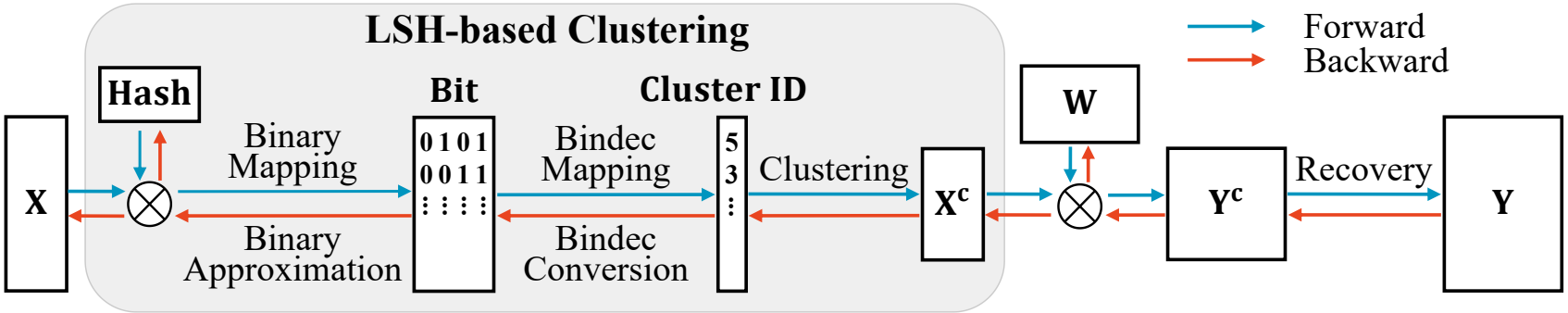


Figure 4: Architecture design of TREC. "⊗" denotes matrix multiplication.

Design

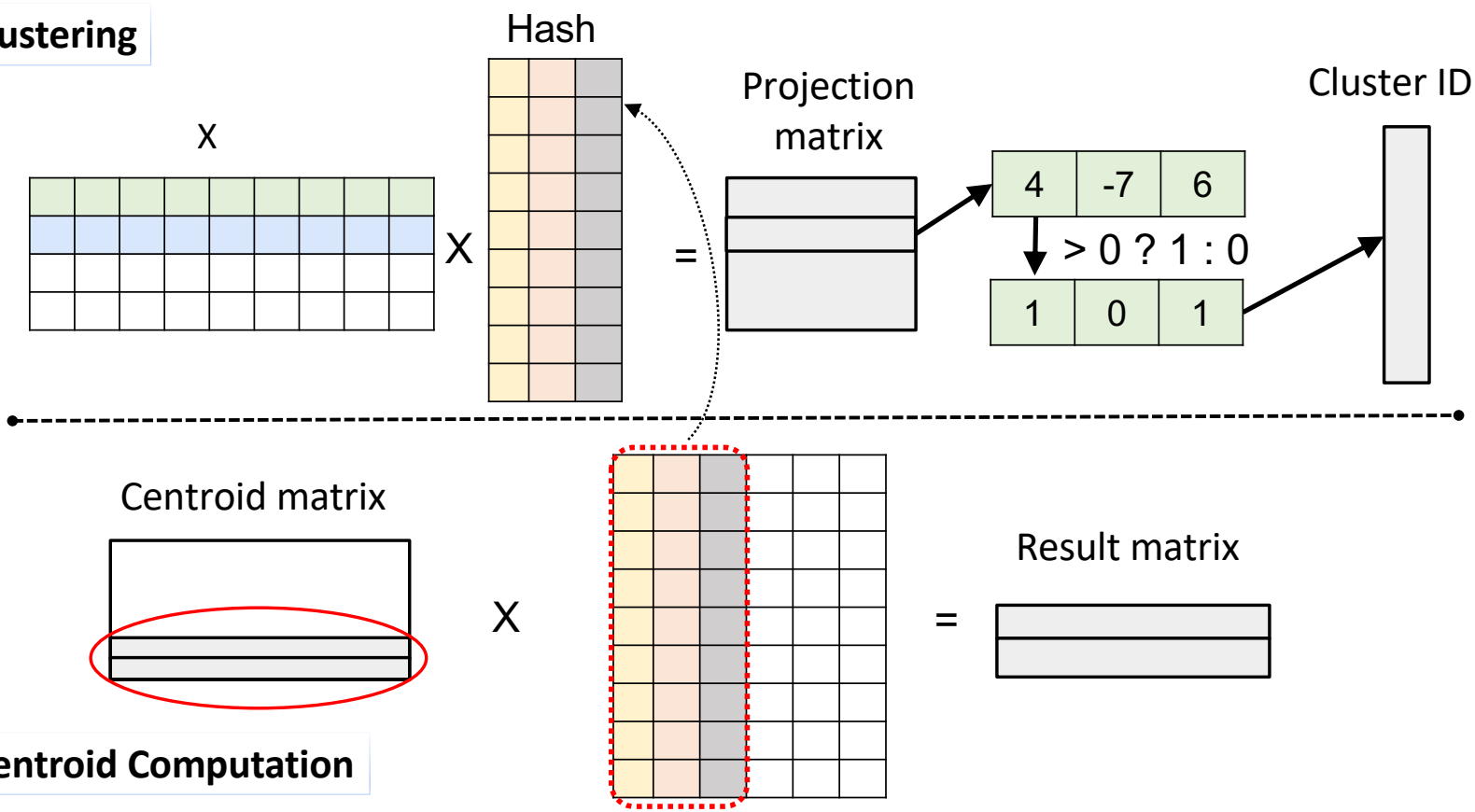


Figure 5: Kernel Reuse Technique.

Design

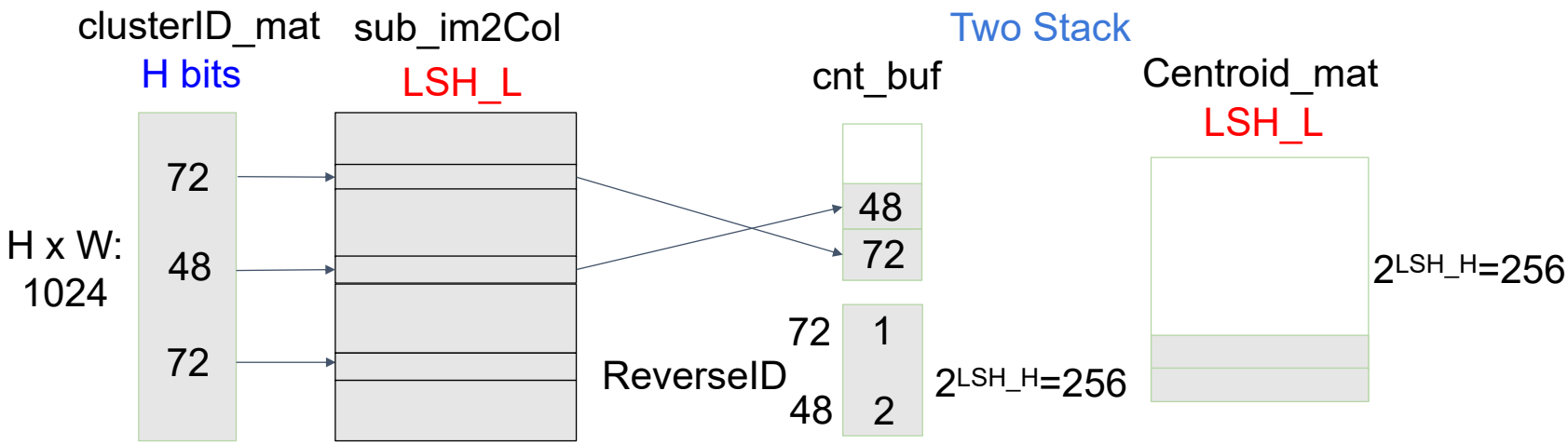


Figure 6: Two-step Stack Technique.

Result

Table 2: End-to-end performance and accuracy comparison.

Target(s)	Convolution Methods	CifarNet	ZfNet	SqueezeNet	SqueezeNet (Bypass)
STM32F469NI	Latency (ms)-CMSIS Conv	217.32	3557.32	1639.51	1998.86
	Latency (ms)-*Deep reuse*	154.44	814.03	—	—
	Latency (ms)-TREC	153.92	814.01	327.9	543.71
	Speedup-TREC vs. CMSIS Conv.	1.412×	4.37×	4.98×	3.68×
STM32F746ZG	Latency (ms)-CMSIS Conv	120.62	1758.73	894.16	1152.46
	Latency (ms)-*Deep reuse*	98.44	525.03	—	—
	Latency (ms)-TREC	97.79	524.3	181.49	274.2
	Speedup-TREC vs. CMSIS Conv.	1.23×	3.35×	4.93×	4.20×
Both	Accuracy (%)-CMSIS Conv.	78.2	80.1	83.5	85.6
	Accuracy (%)-*Deep reuse*	73.2 $\sim$ 76.1	72.5 $\sim$ 76.6	79.8 $\sim$ 81.9	80.5 $\sim$ 83.1
	Accuracy (%)-TREC	76.5	78.9	83	85.3

Opportunities

- Multiple tiles in a channel of an image may be similar to one another.
- All the previous explorations on reuse, however, have been based on a single, most straightforward pattern.

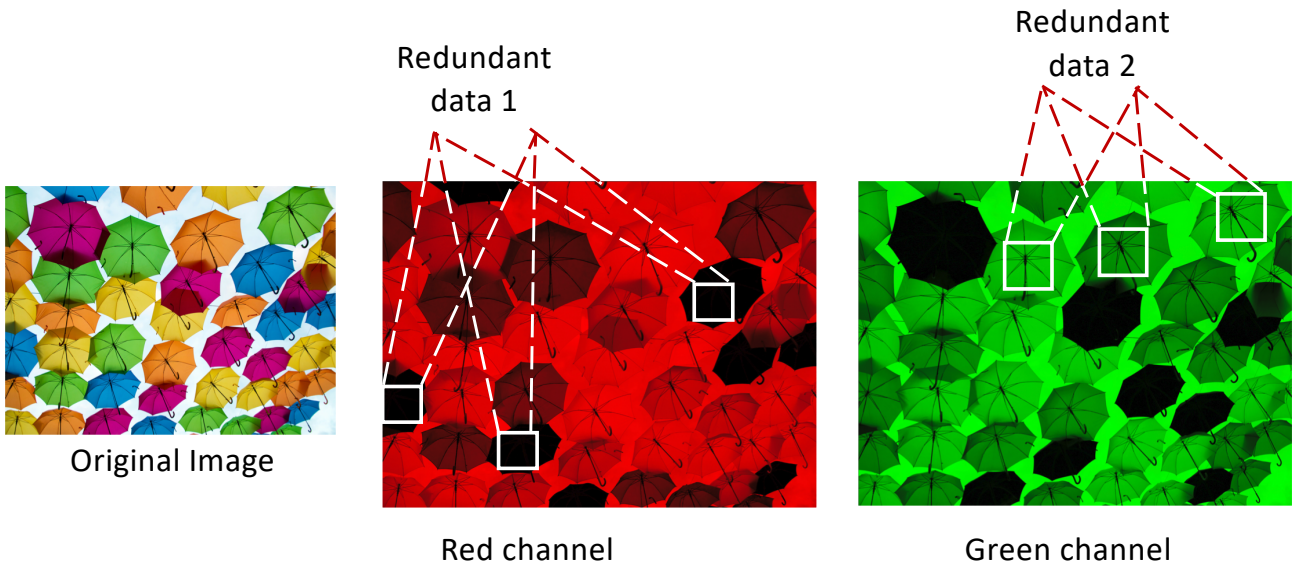


Figure 7: Illustration of similar tiles in an image in each of its channels (only two channels are shown).

Reuse Patterns

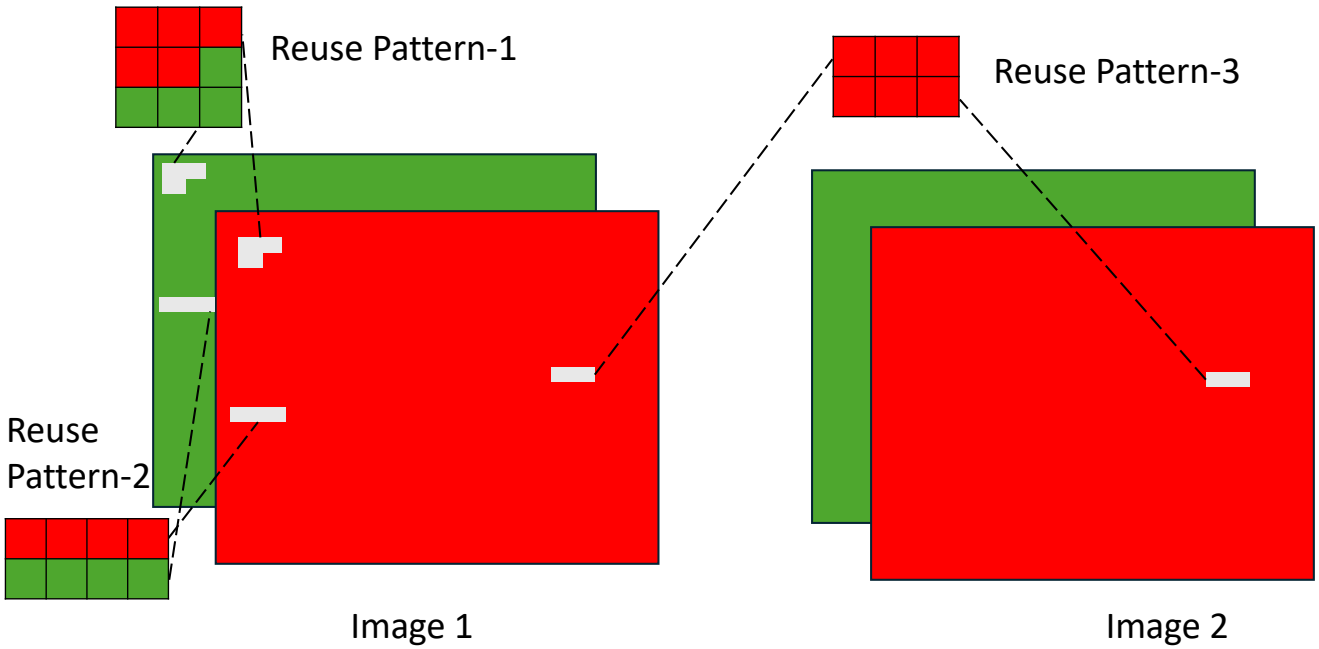


Figure 8: Reuse patterns.

Generalized Reuse: Three Views and Reordering

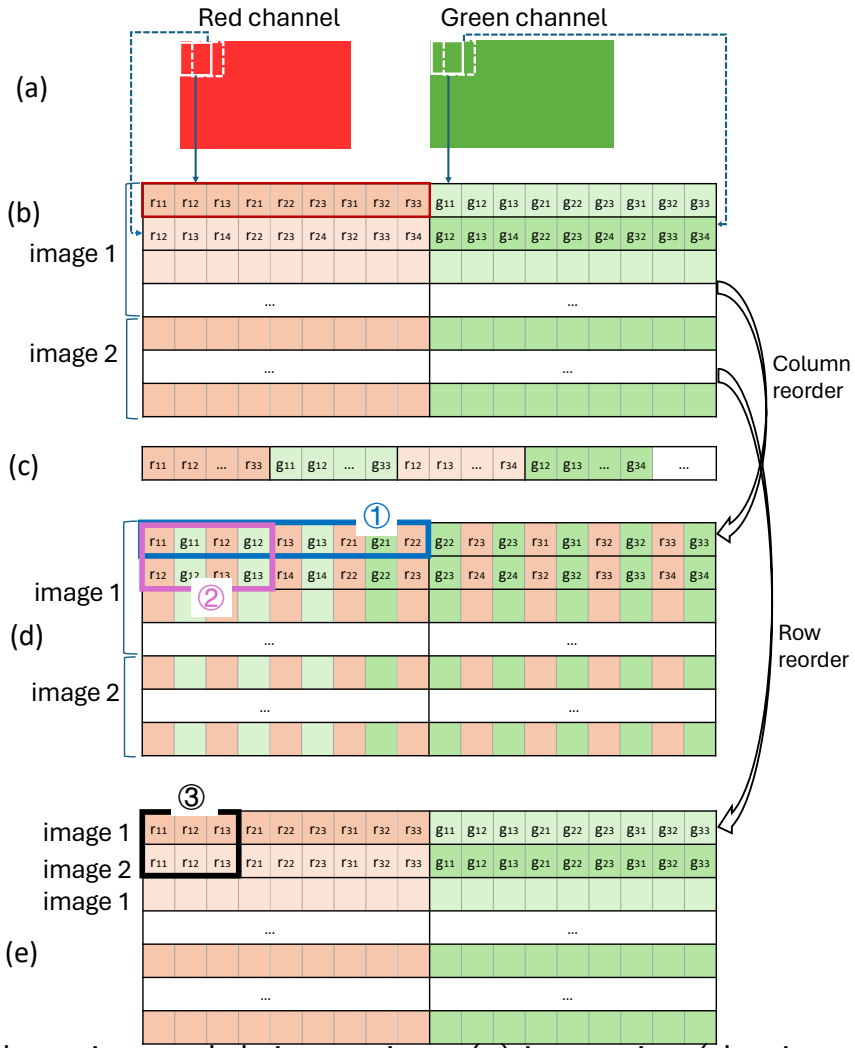


Figure 9: Example of three views and their mappings. (a) image view (showing only two channels); (b) im2col view (default); (c) memory view derived from part b (row major); (d) im2col view after a column reorder; (e) im2col view after a row reorder.

Workflow for Reuse Pattern Selection

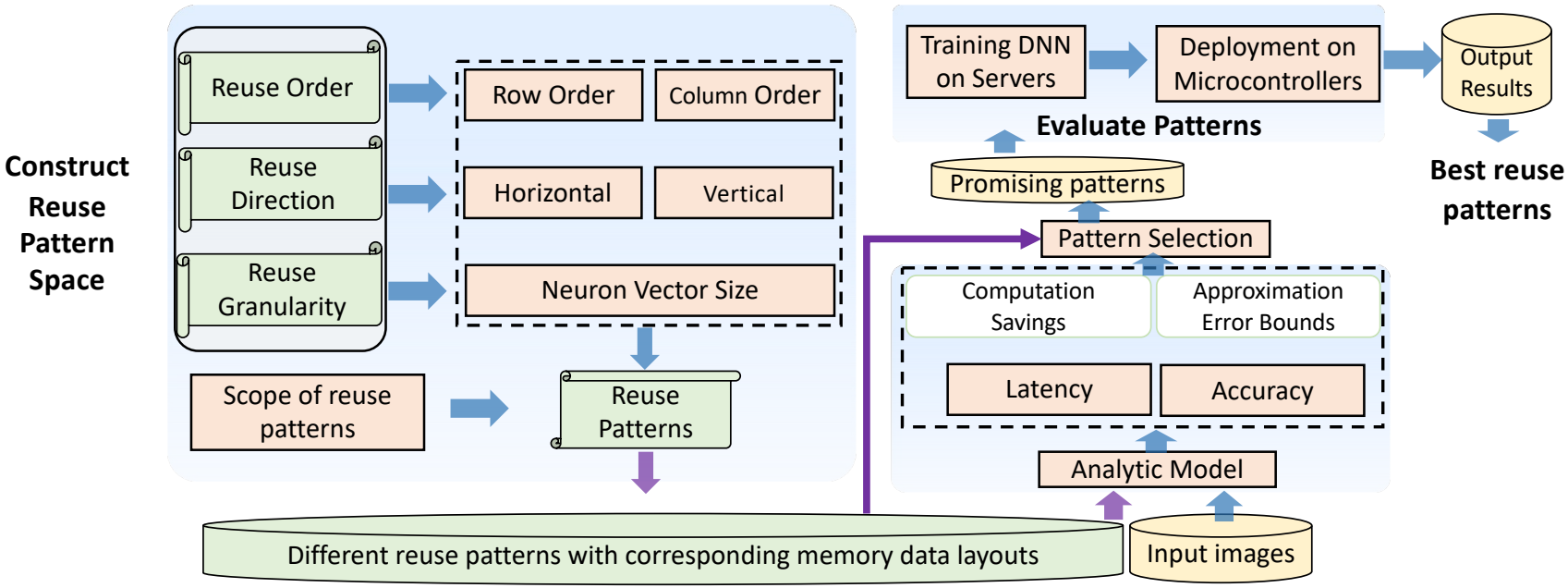


Figure 10: The workflow of analytical-empirical combined approach to reuse pattern selection.

Results

- The approach delivers 1.03-2.2× speedups or 1-8% accuracy improvement over conventional reuse.

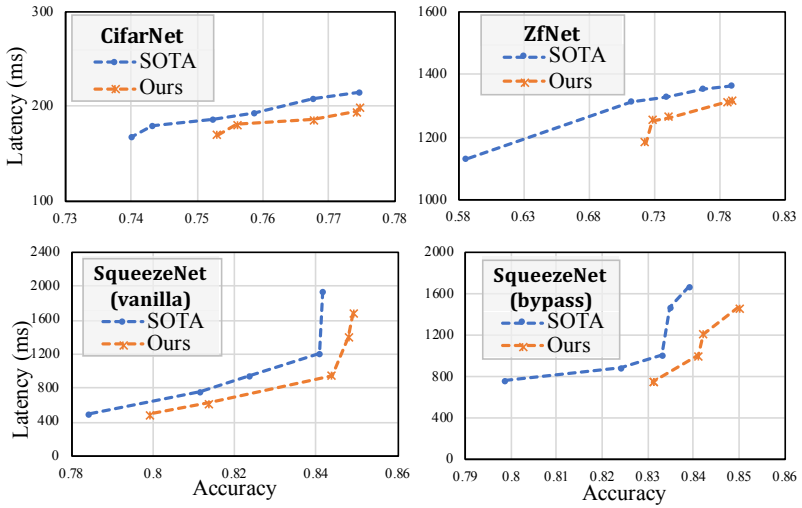


Figure 11: End-to-end results for different networks on STM32F4.

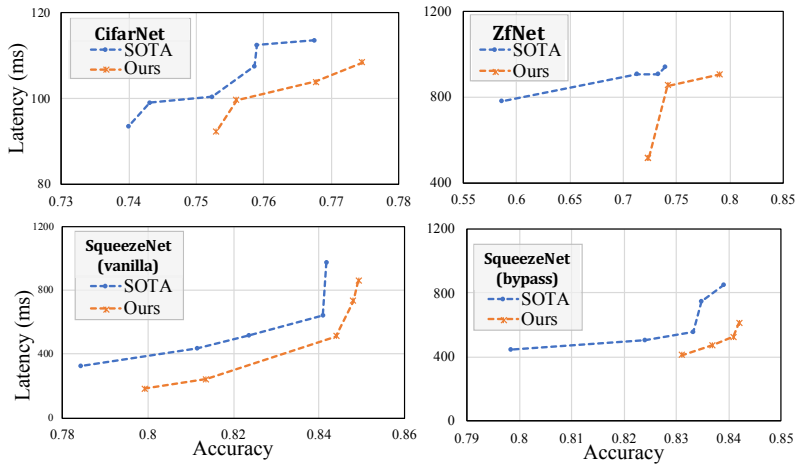


Figure 12: End-to-end results for different networks on STM32F7.

Problem Motivation

Early stopping and multi-fidelity based models are helpful for achieving efficiency in hyperparameter optimization (HPO) problems.

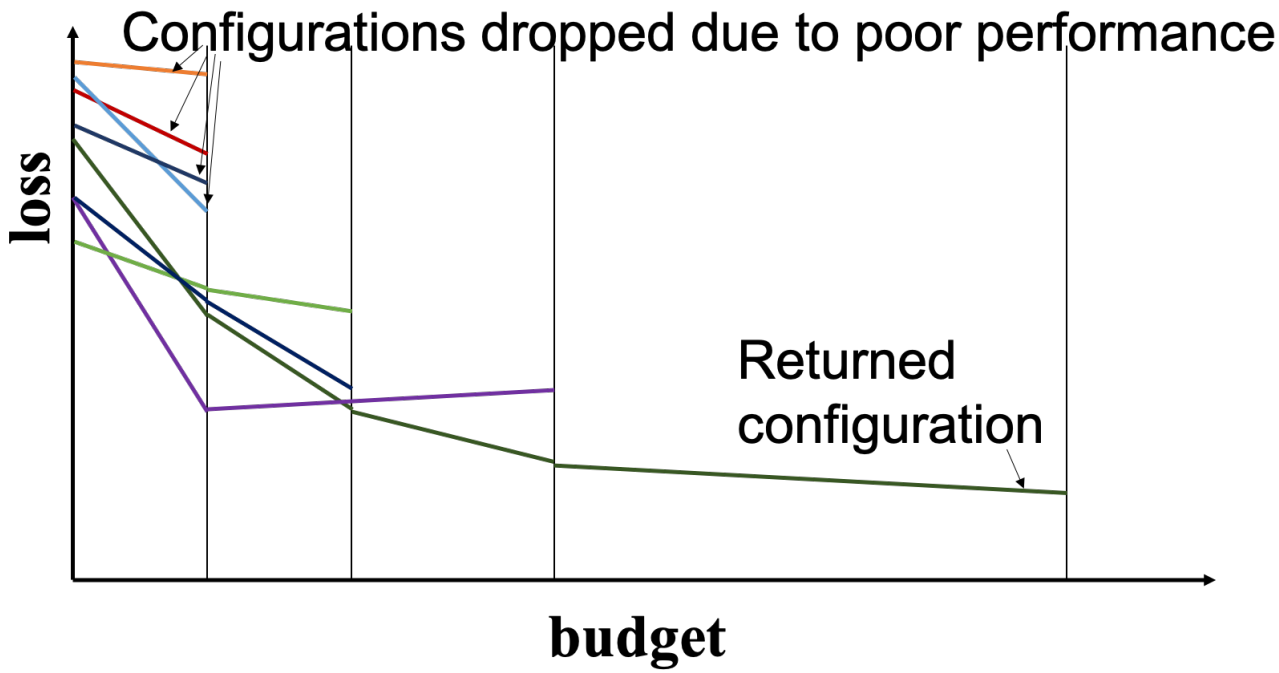


Figure 13: Illustration of Successive Halving (SH) method on different configurations (indicated by different colors).

Problem Motivation

But they lack systematic treatment of uncertainty inherent in the dynamics of the training process of iterative machine learning applications.

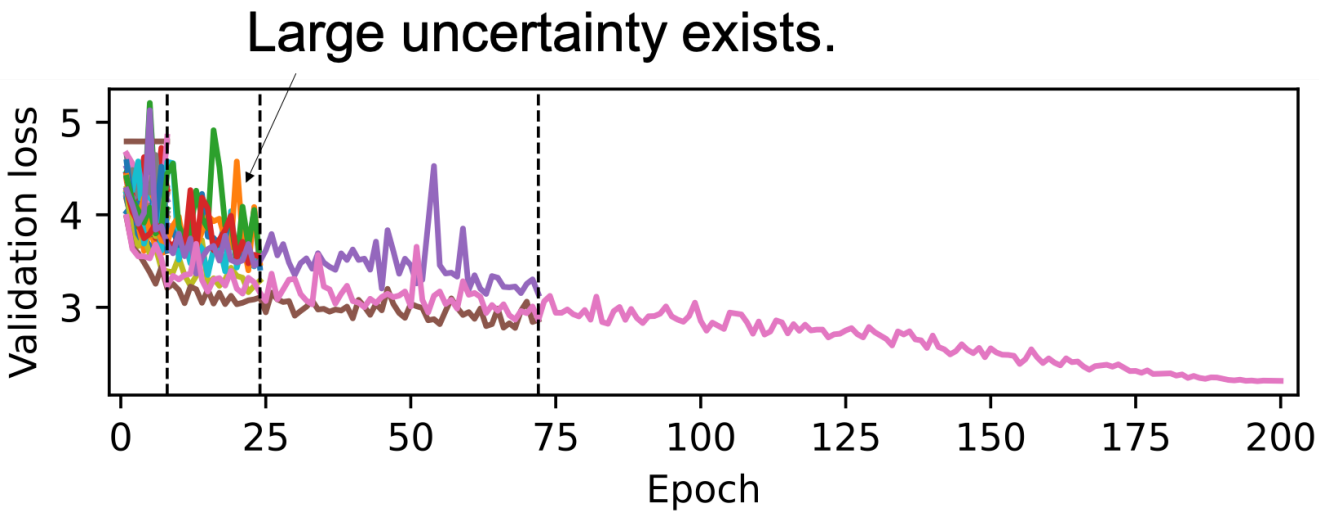


Figure 14: One real SH round.

Design

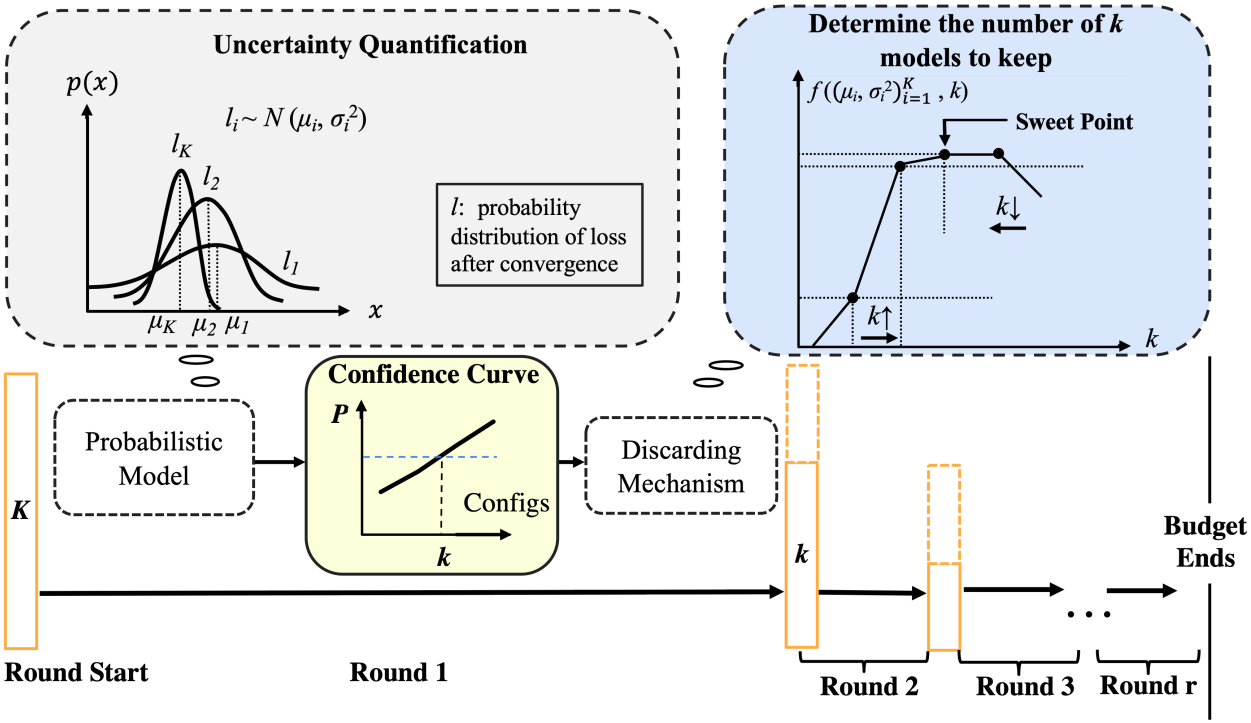


Figure 15: Work flow of UQ-guided HPO. Sweet point is determined by calculating $f((\mu_i, \sigma_i^2)_{i=1}^K, k)$, which captures the effects of keeping k top candidates $(1 \leq k \leq K)$ for the next round, by considering the tradeoff between the risks of discarding the best candidate and the training budget each top candidate can get.

Results

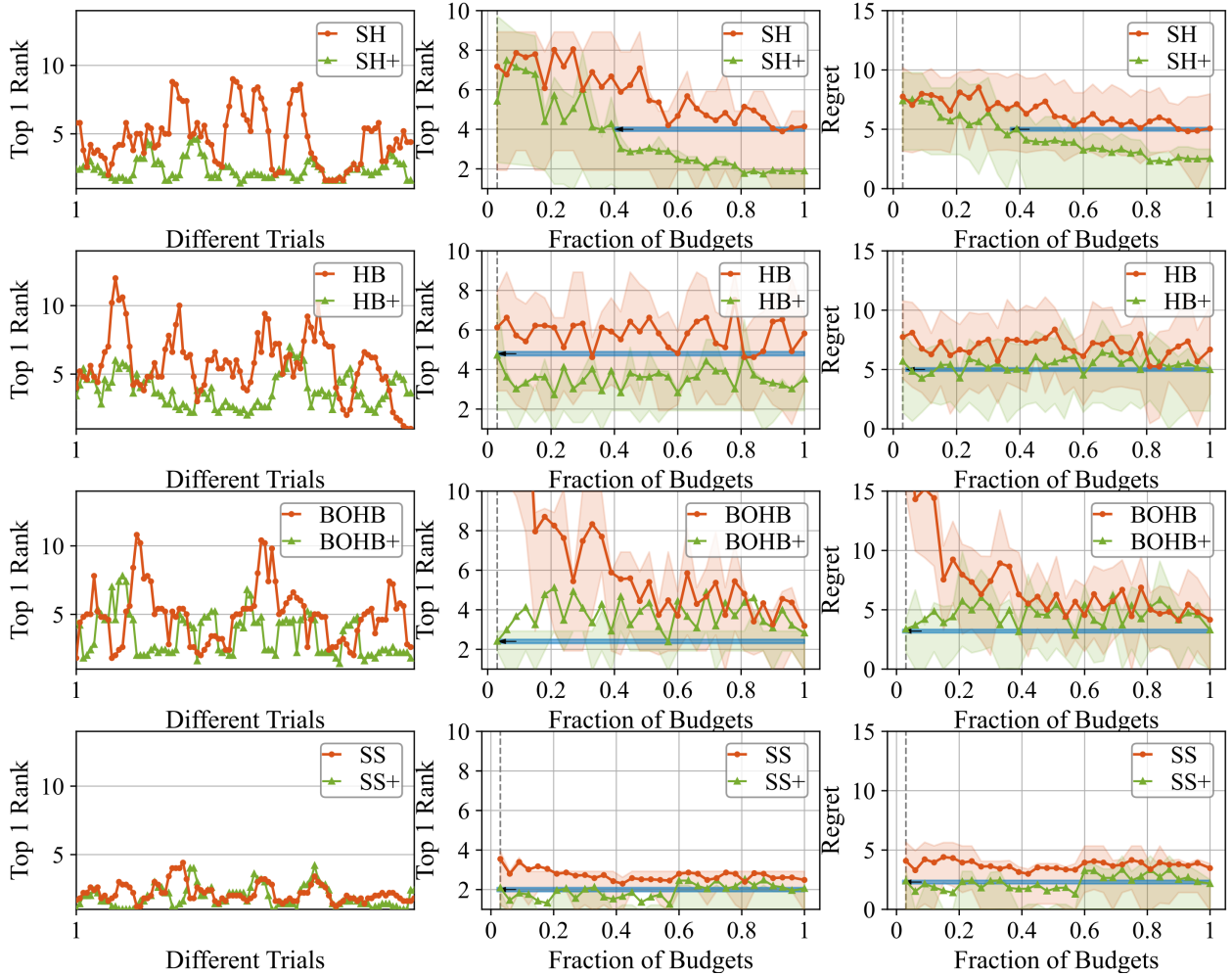


Figure 16: Experimental results of UQ-oblivious HPO methods and their UQ-guided enhancements on NAS-BENCH-2.0

Background

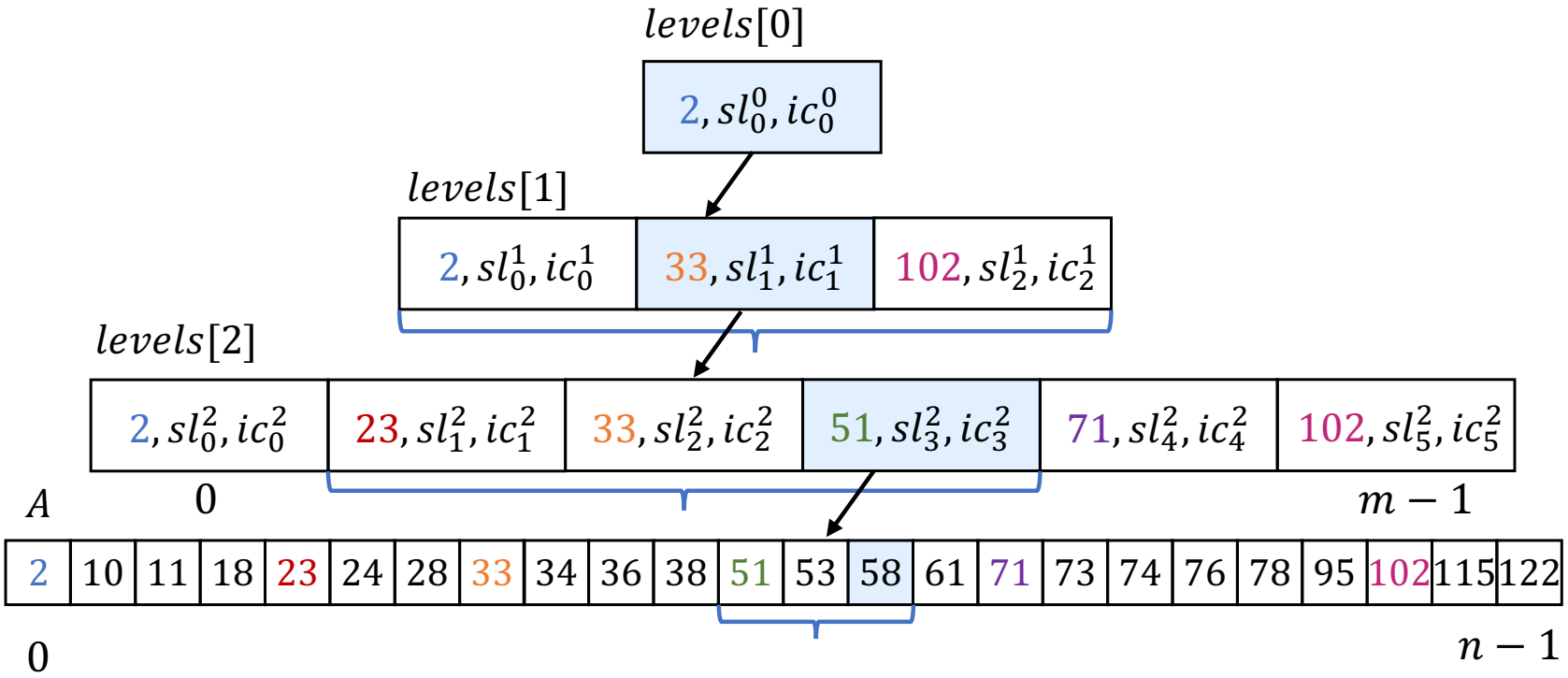


Figure 17: A recursive search process of $x = 58$ in a three-level PGM Learned index with $\epsilon = 1$.

We aim to accelerate the learned indexes on GPUs and design a heterogeneous model.

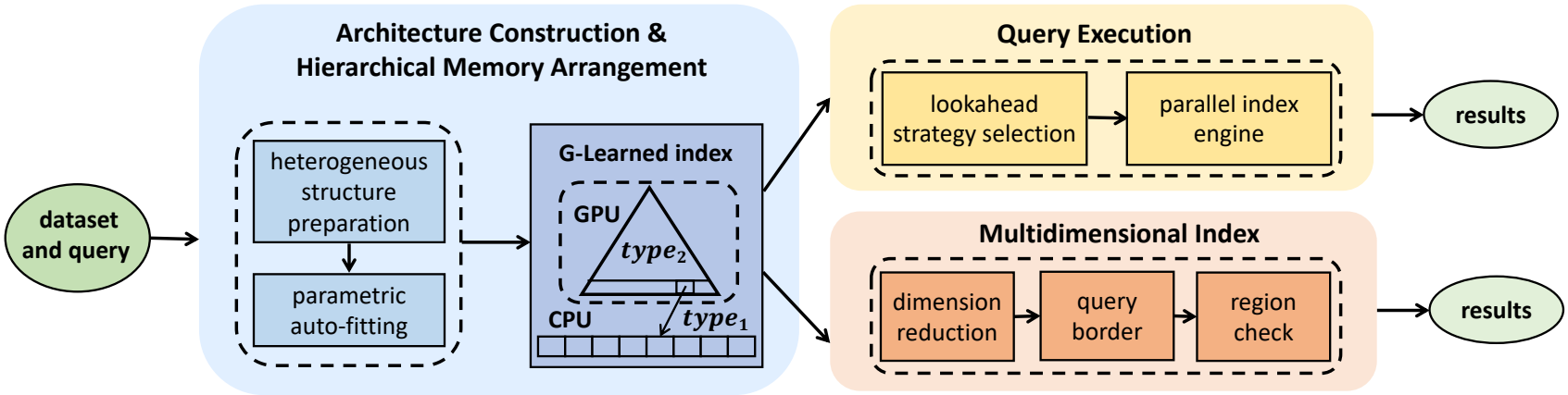
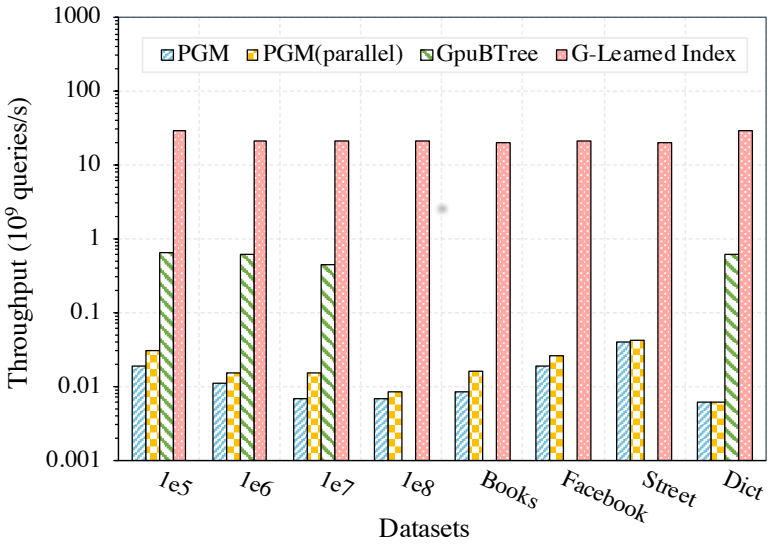


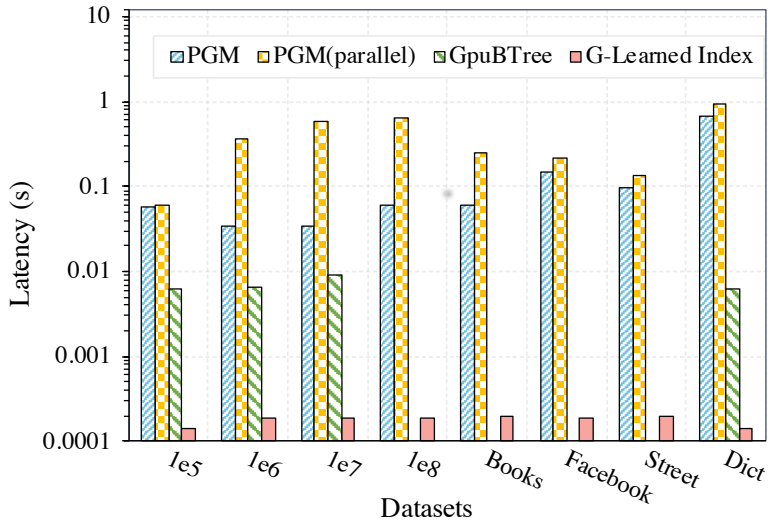
Figure 18: Overview of the G-Learned index. There are four components: architecture construction, hierarchical memory arrangement, query execution and multidimensional index.

G-Learned Index

- Our method can accelerate the state-of-the-art learned indexes.
- We also achieve satisfactory improvements on range queries.



(a) Throughput



(b) Latency

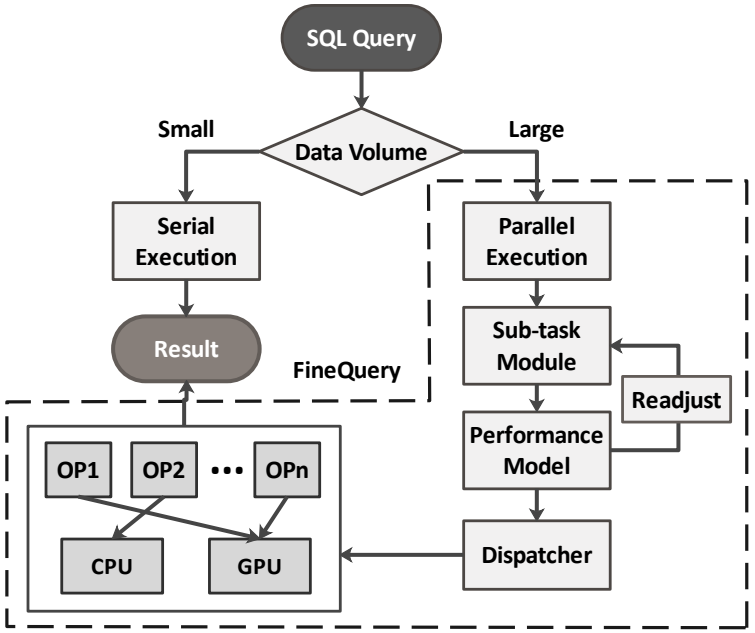
Figure 19: Performance of different methods.

Background

- Huge amounts of data generated at the edge with limited computing capacity.
- Integrated heterogeneous co-processors into edge devices.

Contribution

- Utilize the unified memory by enabling the CPU and GPU to access memory in a suitable pattern.
- Propose a performance model combined with statistic data and input data attributes (sub-task, pipeline).
- Develop a sub-task module to split a specified query into several operators.



Results

- FineQuery on the Jetson platform can reduce the latency by 56.69% compared to the query processing on the discrete architecture.

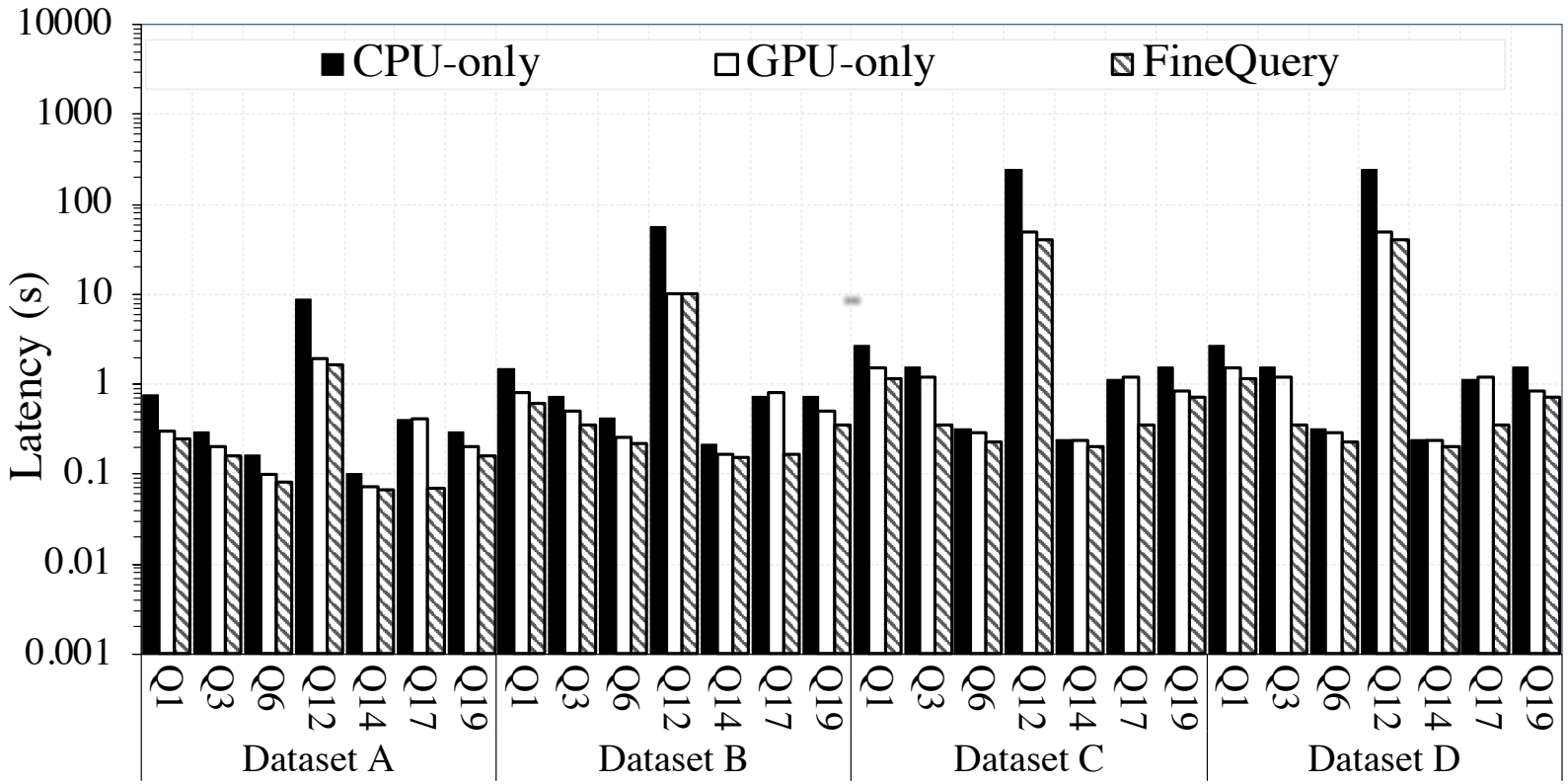
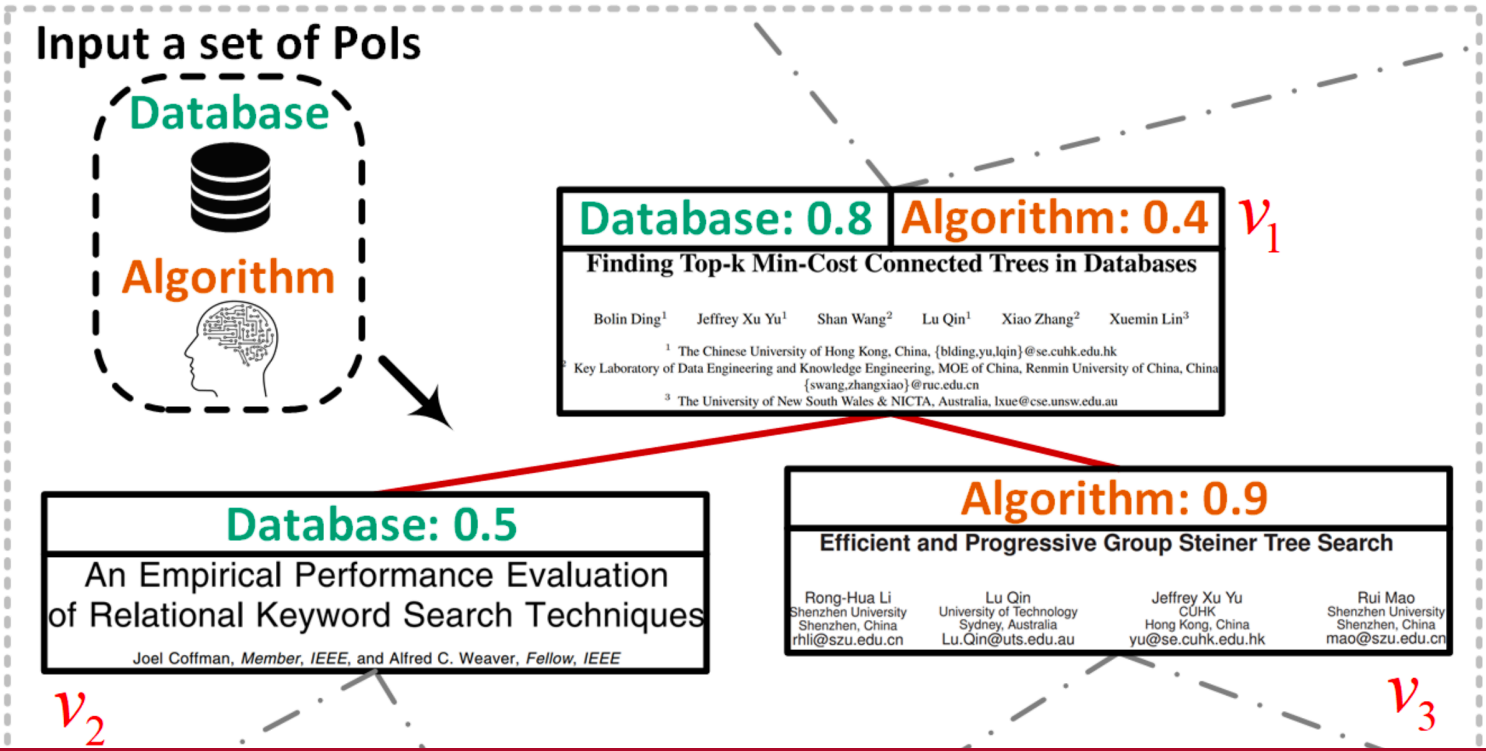


Figure 21: Latency on Jetson AGX Xavier.

Background

- Techniques for Group Steiner Trees are widely used in graph processing applications and data mining.
- Current works are based on assumptions that each node is associated with deterministic property of interest (PoI), ignoring the probablistic scenarios prevalent in ML labeling.



Contribution

- Developed the new algorithm to approximate the solution.
- Devised and implemented the parallel version of pruned landmark labeling algorithm and achieved significant speedups.

Thanks!