AccelOpt: A Self-Improving LLM Agentic System for AI Accelerator Kernel Optimization

Genghan Zhang[1] Shaowei Zhu[2] Anjiang Wei[1] Zhenyu Song[2] Allen Nie[2] Zhen Jia[2]

Nandita Vijaykumar[23] Yida Wang[2] Kunle Olukotun[1]

[1]Stanford University, [2]AWS, [3]University of Toronto

Presenter: Jingwen Sun, Yuhang Wang @ USTC

Outline

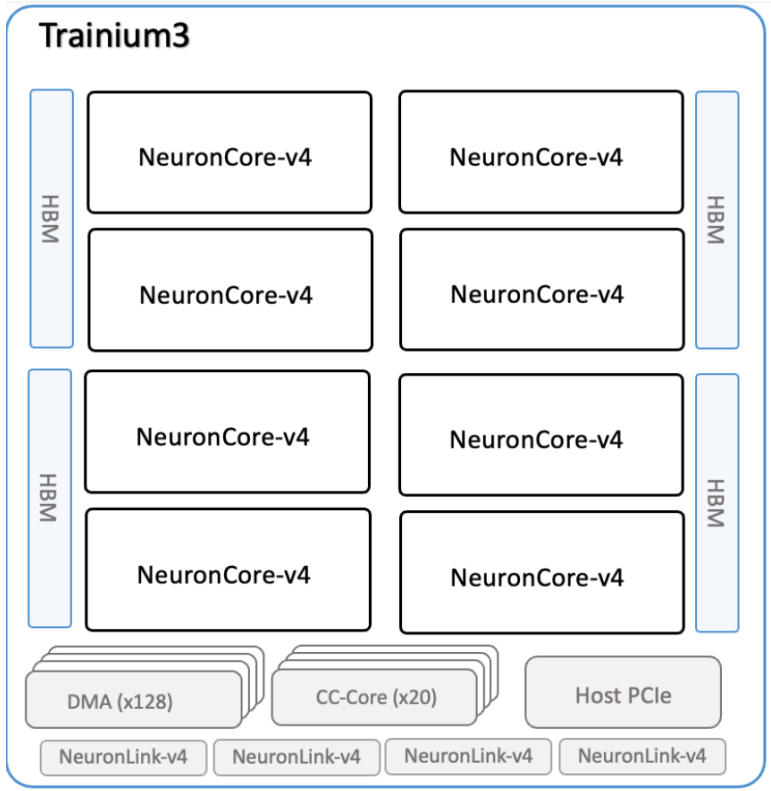
- **Background**
  - AI Accelerator & Kernel Optimization & Agentic Workflow
- **Motivation & Goal**
- **Design**
  - Beam Search
  - Optimization Memory Curation
- **Profiling Service**
- **Evaluation**
- **Conclusion**

Background – AI Accelerator

❑Diverse AI accelerators

- ❖to handle/speed up specific tasks
- ❖require different kernels
- ❖eg. AWS Trainium3

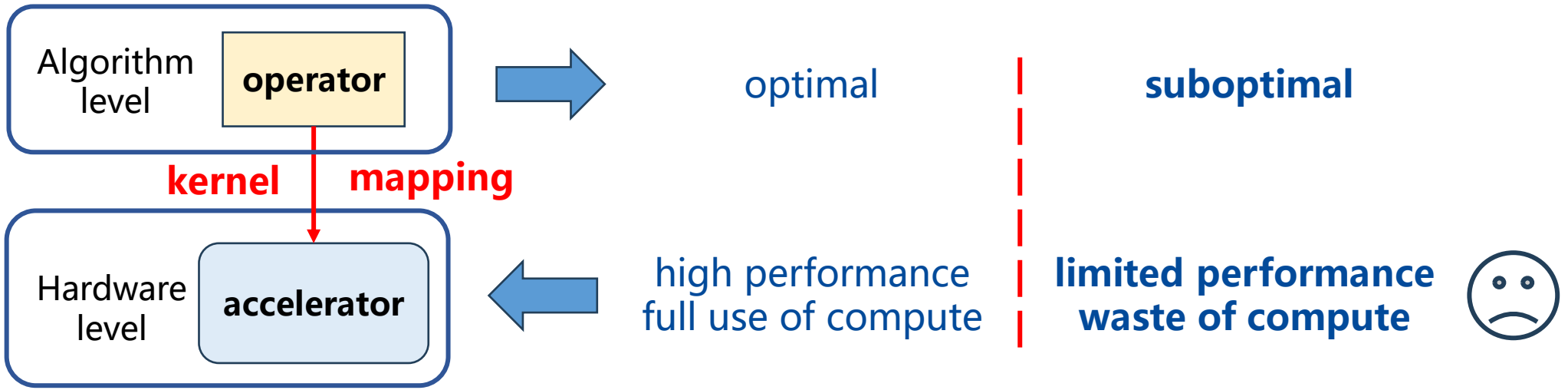
MXFP8	~2,517 TFLOPs
BF16/FP16	~671 TFLOPs
FP32	~183 TFLOPs
HBM Capacity	144 GiB HBM3e
HBM Bandwidth	4.9 TB/s
SBuf	~256 MiB
DMA	4.9 TB/s
Core Arch	8 × NeuronCore-v4 cores



Background – Kernel Optimization

❑Why do we need to optimize kernels?

❖to achieve higher performance



Background – Kernel Optimization

- **Why do we need to optimize kernels?**
  - to achieve higher performance
- **Is optimization easy?** **NO** ☹
  - **huge search space**
    - tiling, fusion, workloads, parallelism, memory…..
    - Prior systems also view optimization as search:
      - TASO(SOSP'19), PET(OSDI'21), Mirage(OSDI'25)

Background – Kernel Optimization

- **Why do we need to optimize kernels?**
  - to achieve higher performance
- **Is optimization easy?** **NO** ☹
  - huge search space
  - **manual cost** → Even on mature GPUs, tuning a high-performance kernel take years

Background – Kernel Optimization

- **Why do we need to optimize kernels?**
  - to achieve higher performance
- **Is optimization easy?** **NO** ☹
  - huge search space
  - manual cost → Even on mature GPUs, tuning a high-performance kernel take years
  - **few opt heuristics** → Heuristics do not transfer directly across accelerators

Background – Agentic Workflow

- **LLMs can help generate kernels**
  - **success case** on GPU such as Autocomp(arXiv)
  - **scale search** beyond manual tuning
  - support **agent cooperation**

Background – Agentic Workflow

❑LLMs can help generate kernels

- ❖success case on GPU such as Autocomp(arXiv)
- ❖scale search beyond manual tuning
- ❖support agent cooperation

❑Current research

Name	Research Focus	Common problem
KernelBench	ML operator → AI accelerator kernel translation	1. Cannot **self-improve** 2. Optimize with **human intervention** 3. Cannot cover all **possible tuning paths**
Autocomp	Optimizes unfused kernels on GPU and Trainium	
AlphaEvolve	Optimizes matmul / FlashAttention kernels on TPU	
GEPA	Optimizes LLM-generated kernels on TPU	

Motivation & Goal

❑Motivation:

- ❖Agentic system for kernel optimization, which:
  - ➢ can search, generate, evaluate and refine kernels **automatically**
  - ➢ perform well on **different accelerators**

Motivation & Goal

❑Motivation:

- ❖Agentic system for kernel optimization, which:
  - ➢ can search, generate, evaluate and refine kernels **automatically**
  - ➢ perform well on **different accelerators**

❑Goal:

① ❖Balance search space coverage with cost efficiency

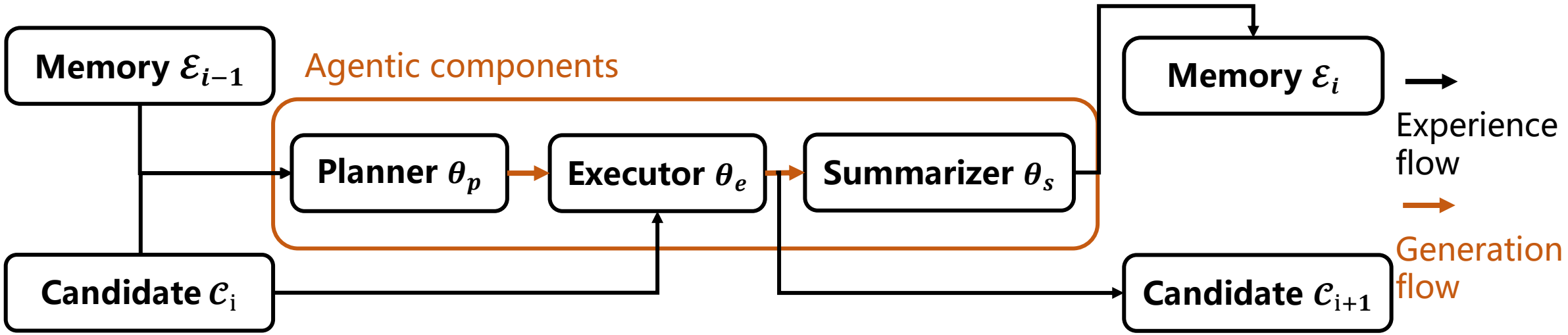
② ❖Avoid manual intervention and best-practice injection

❖Accumulate experience → self-improveing system

Can agents become a kernel engineer on an accelerator it hasn't met before?

Design: Overview

❑Agentic workflow



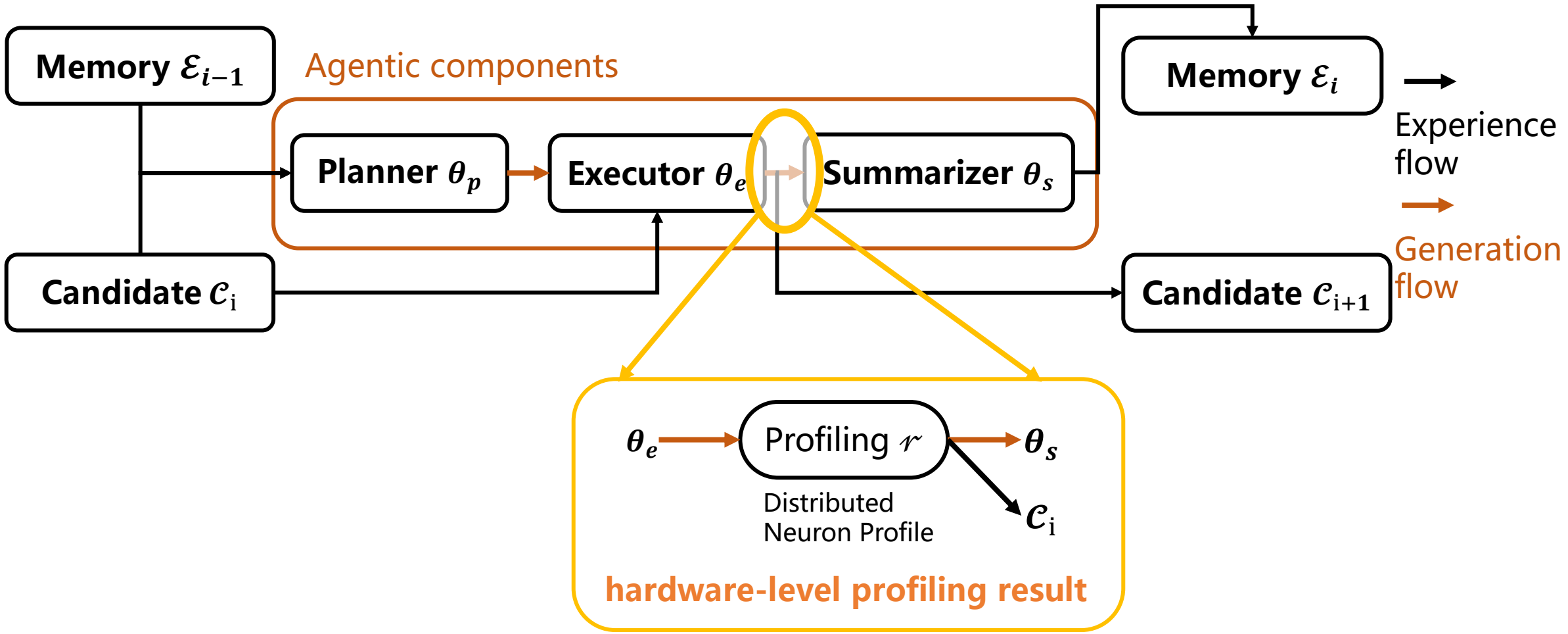
θ_p : **Propose** 1-step opt **plan** based on kernel profiles

θ_e : **Transform** opt **plans** into executable kernel rewrites

θ_s : **Summarize** opt outcomes into reusable **experience**

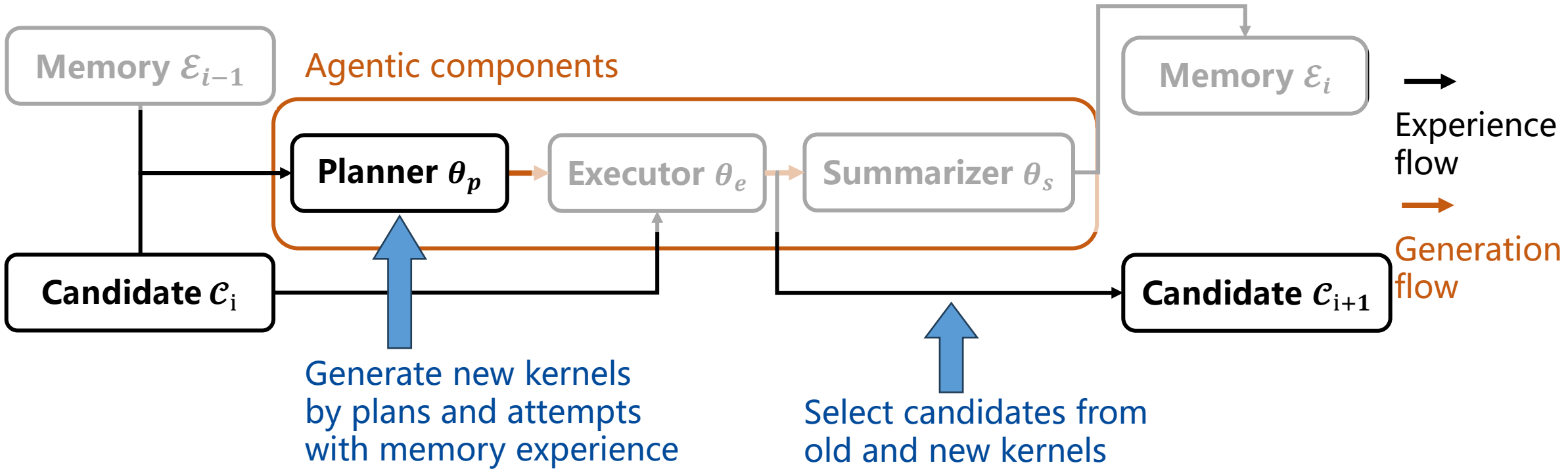
Design: Overview

❑Agentic workflow



Design: Overview

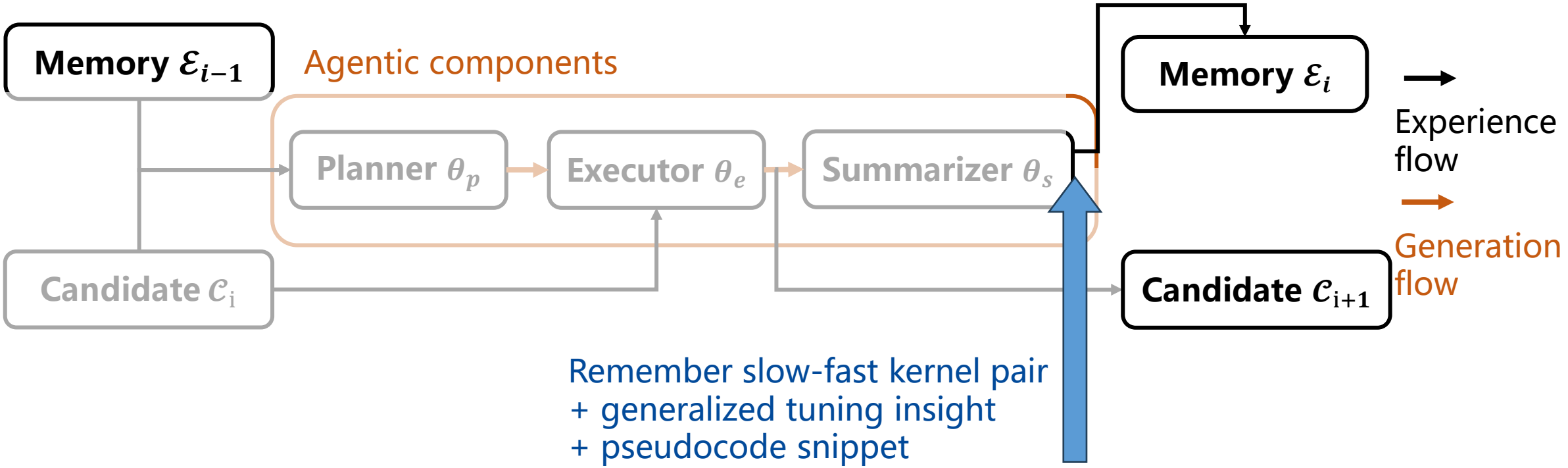
❑Agentic workflow



① Beam Search

Design: Overview

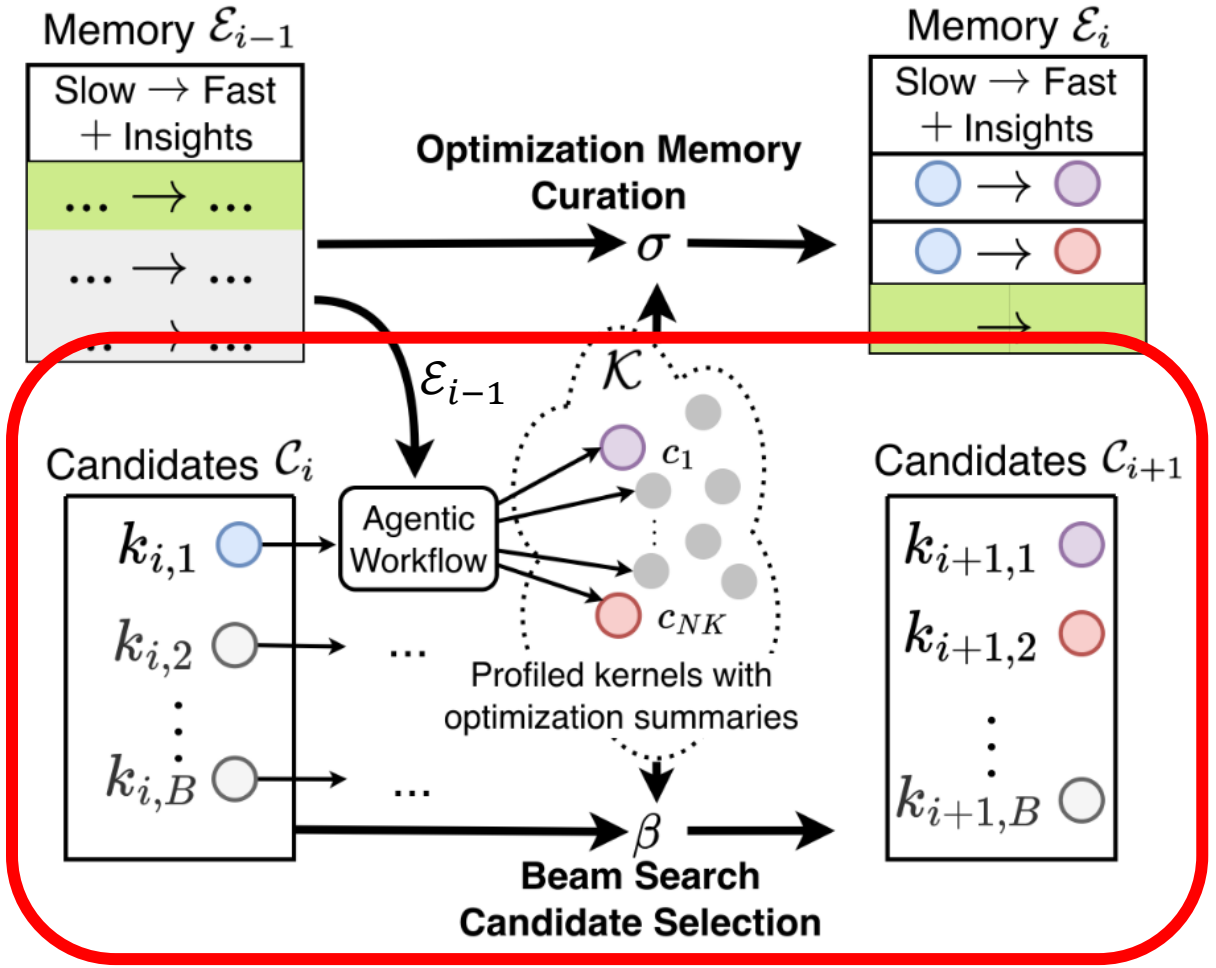
❑Agentic workflow



② Optimization Memory Curation

Design: Beam Search

❑Candidate kernel frontier expansion (program-level)



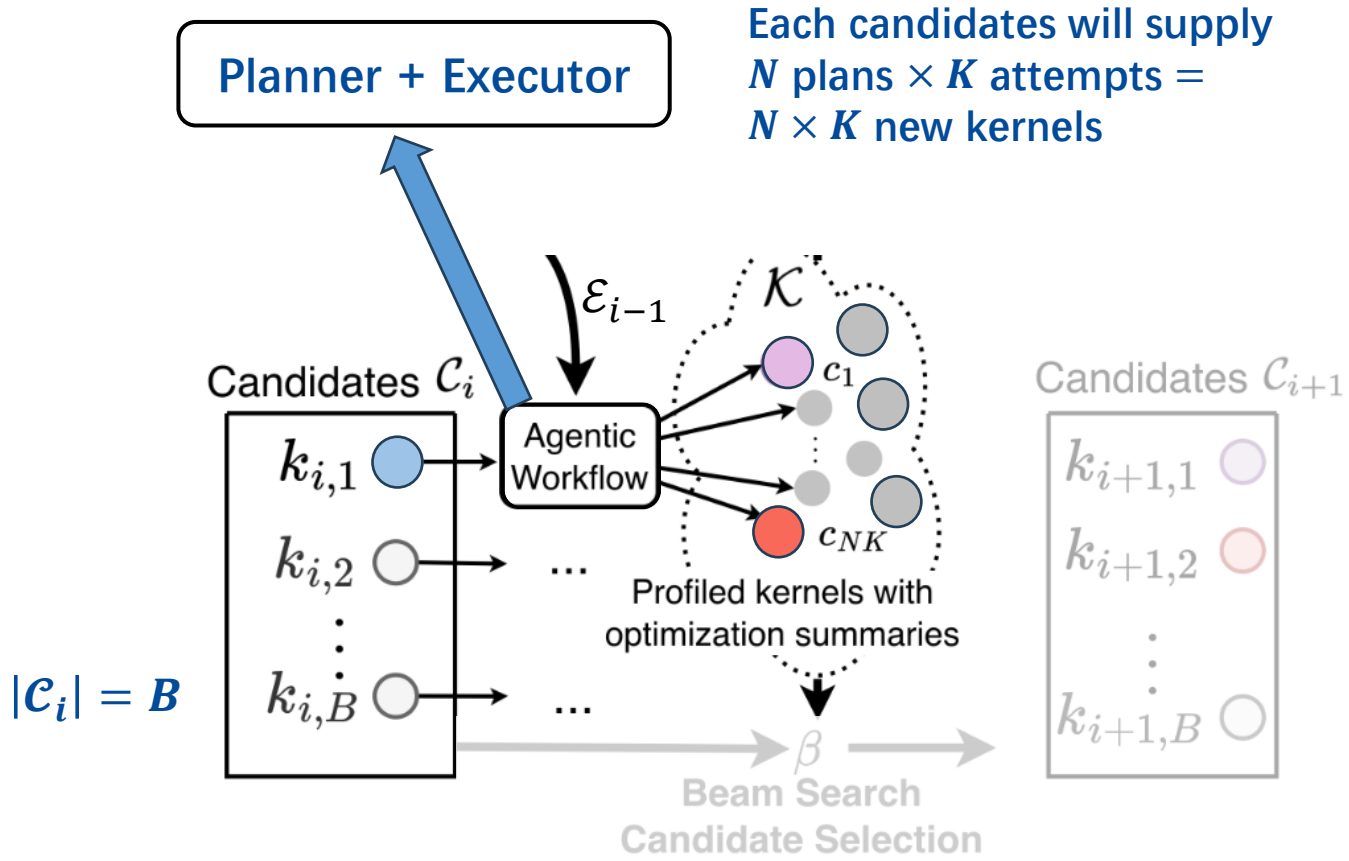
Symbols:

$\mathcal{C}$: set of candidate kernels
$\mathcal{E}$: set of experience
$\mathcal{K}$: set of profiled new-generated kernels

β: candidate selection function
σ: optimization memory curation function

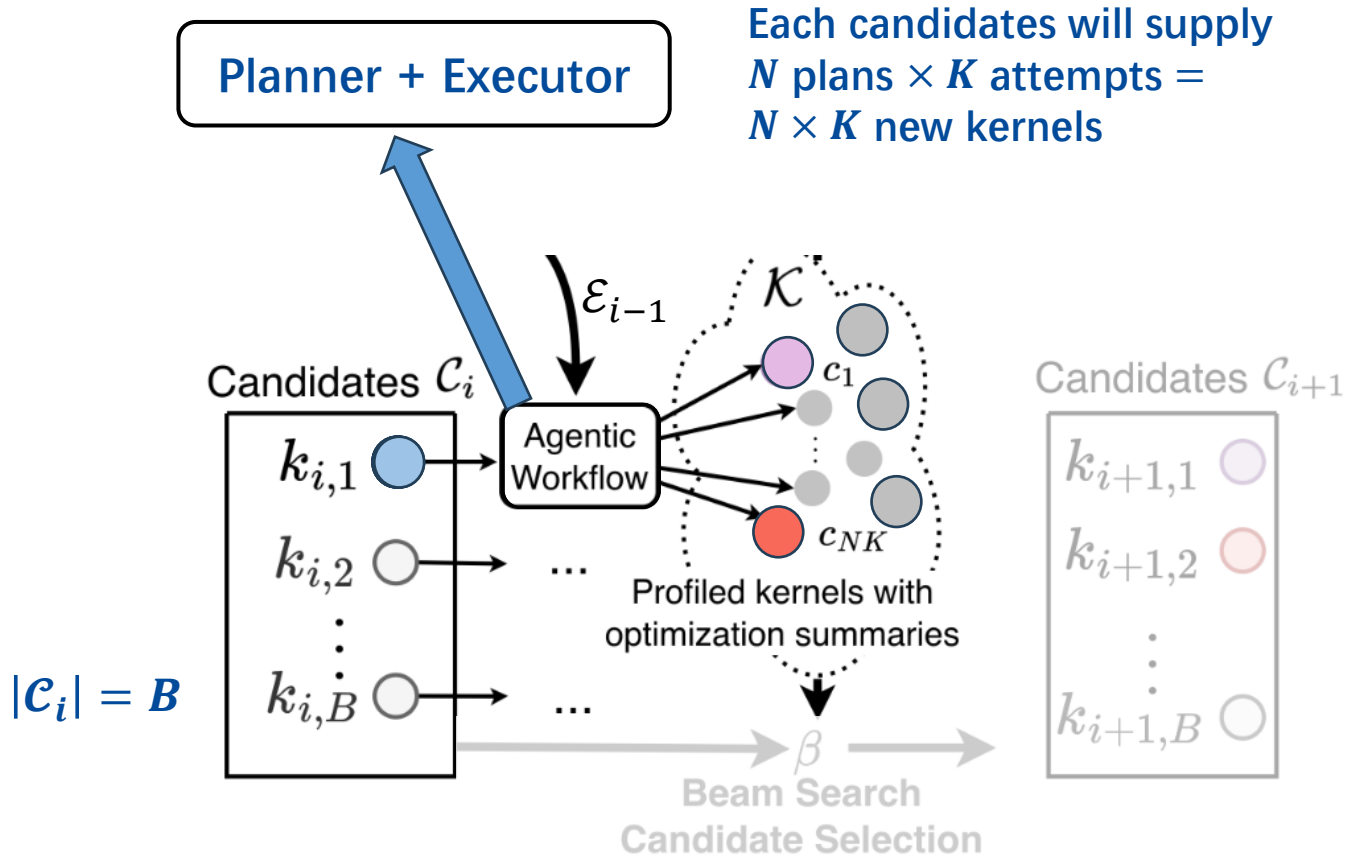
Design: Beam Search

❑Candidate kernel frontier expansion (program-level)



Design: Beam Search

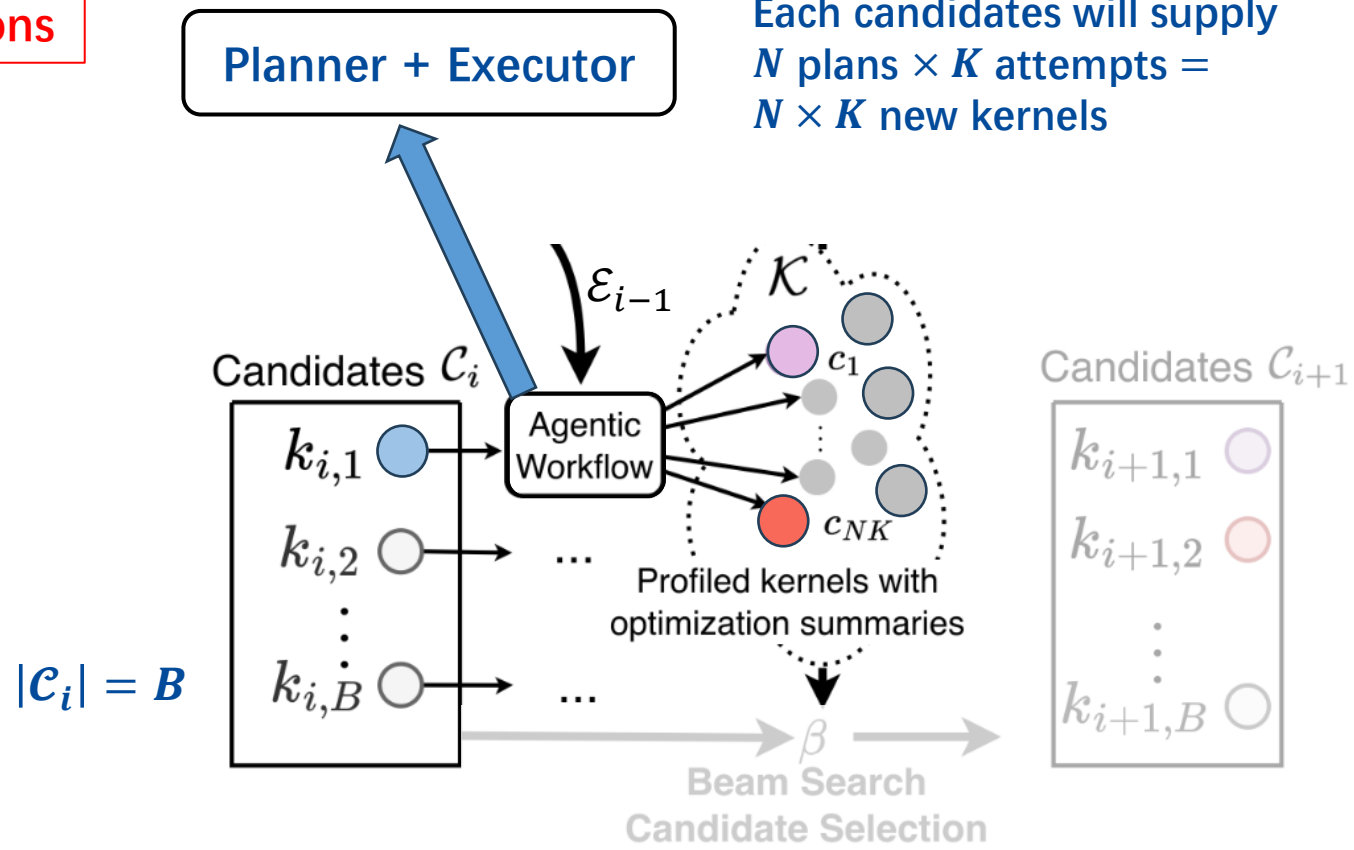
❑Candidate kernel frontier expansion (program-level)



Design: Beam Search

❑Candidate kernel frontier expansion (program-level)

Explore different tuning directions



Design: Beam Search

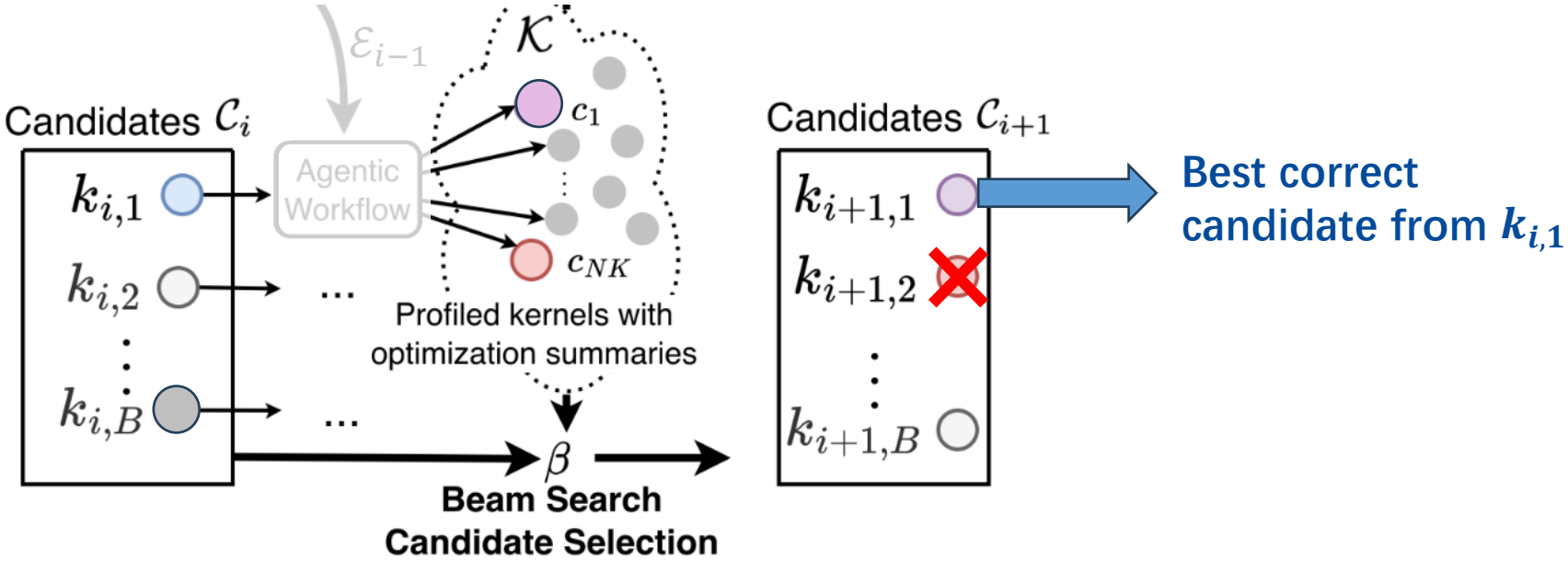
❑Candidate Selection Function β

Reserve promising kernels with diversity

Old candidates and new generated kernels

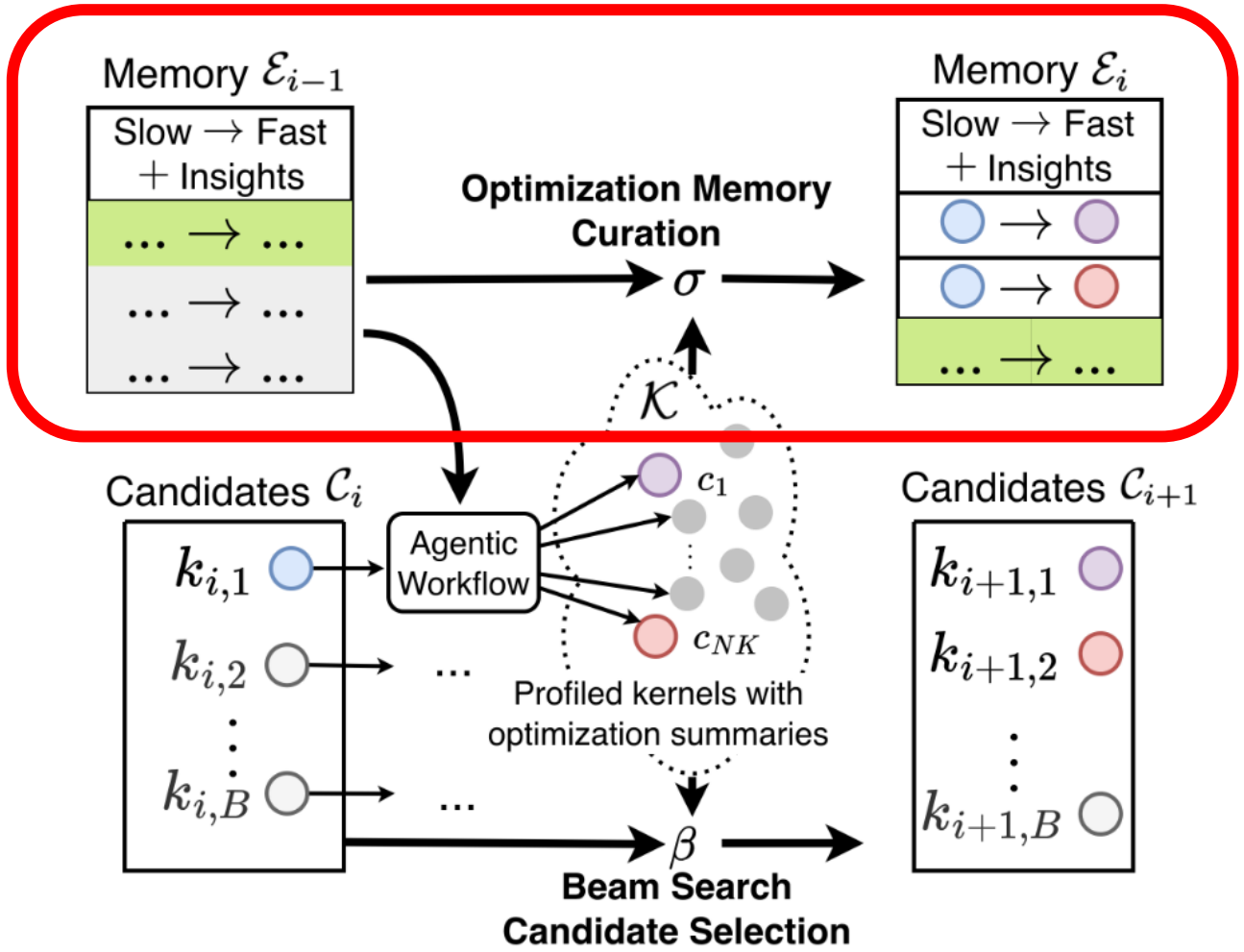
Trade-off: Sampling & Profiling cost

1. Select top-B candidates from $B+B\times N\times K$ kernels
2. If new candidates is less than B: fill slots with $\mathcal{C}_i$



Design: Optmization Memory Curation

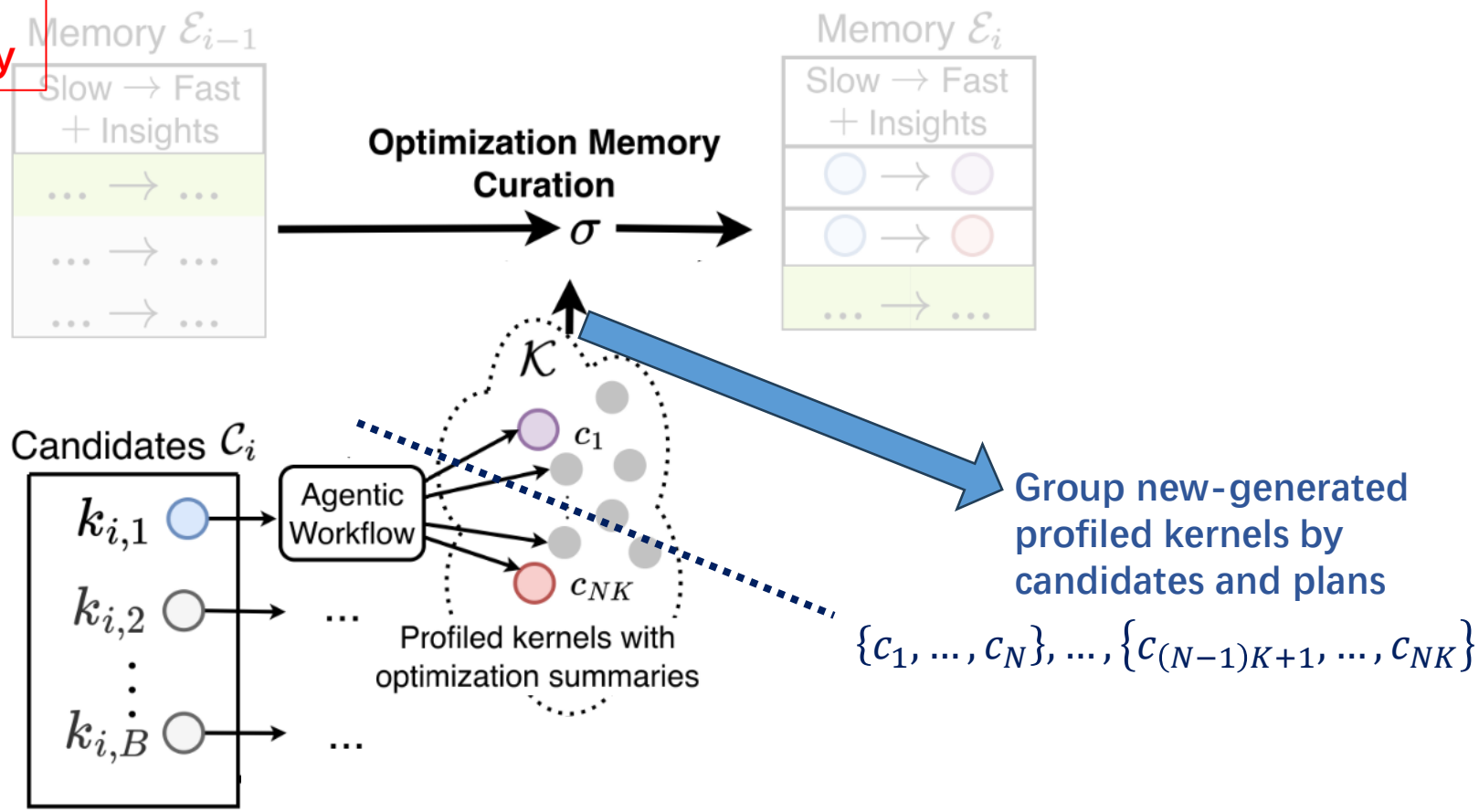
- Capture optimization experience during exploration



Design: Optmization Memory Curation

❑Capture optimization experience during exploration

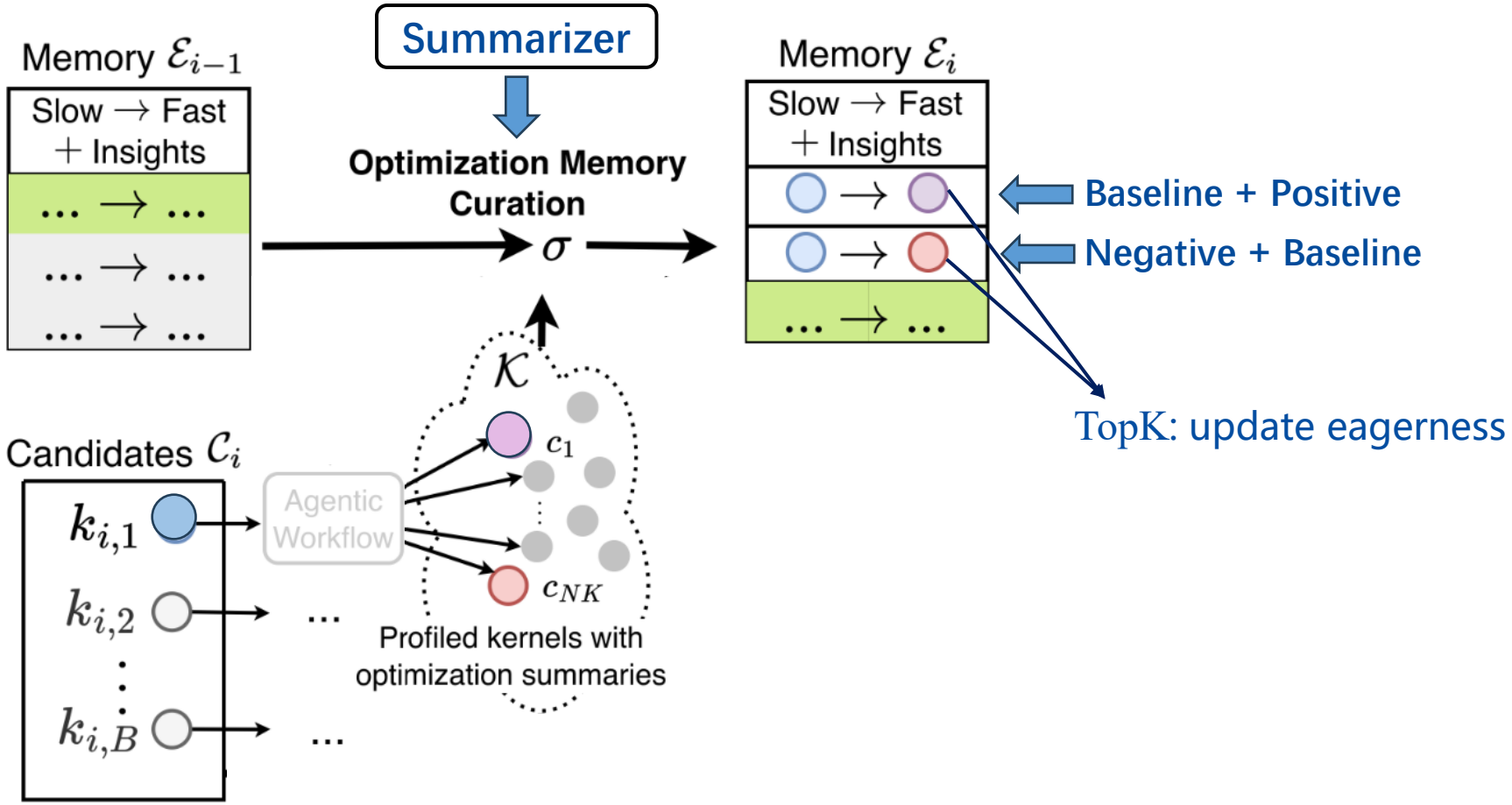
Reserve promising kernels with diversity



Design: Optmization Memory Curation

❑Capture optimization experience during exploration

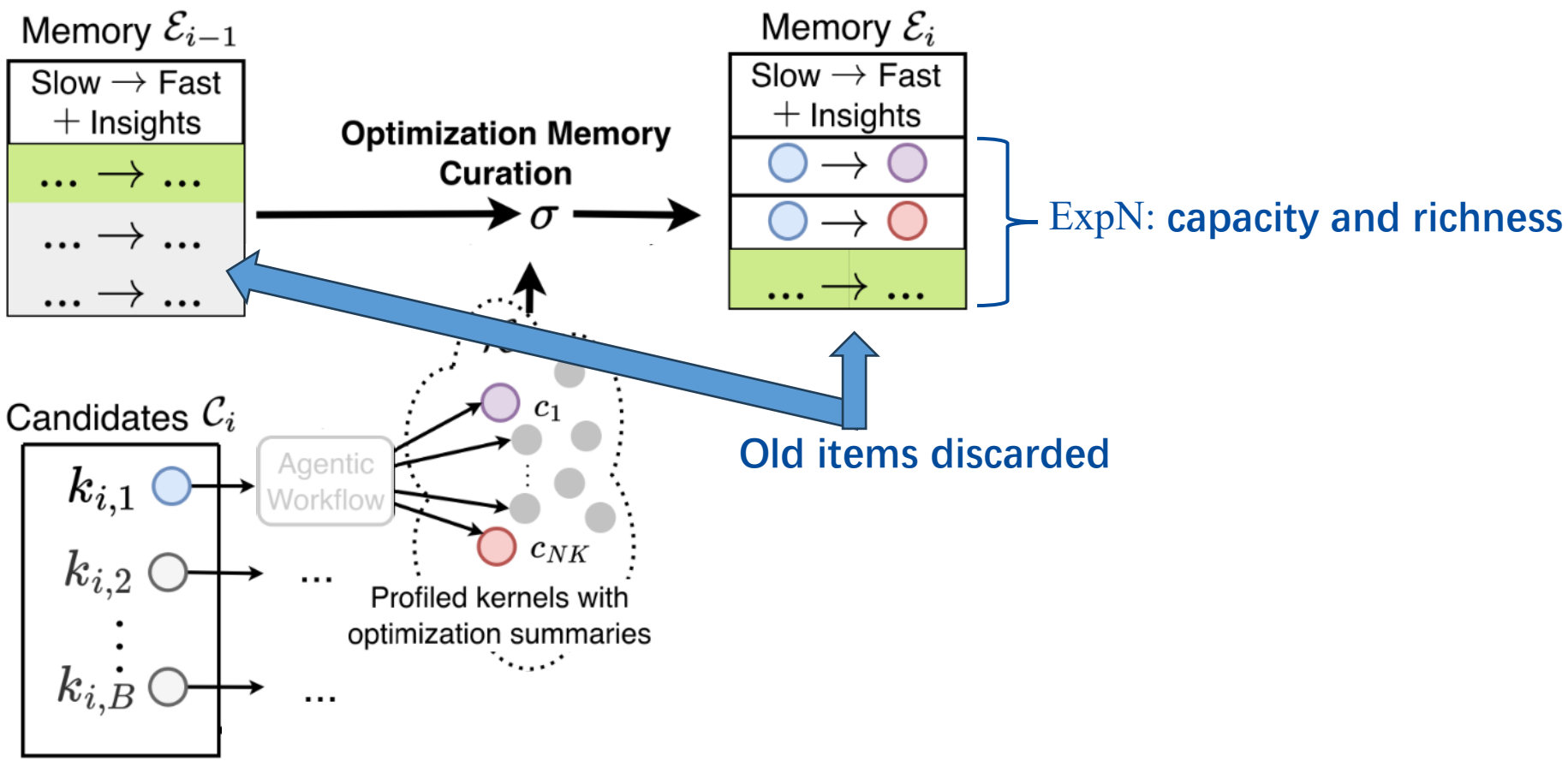
Embrace both successful and failed attempts



Design: Optmization Memory Curation

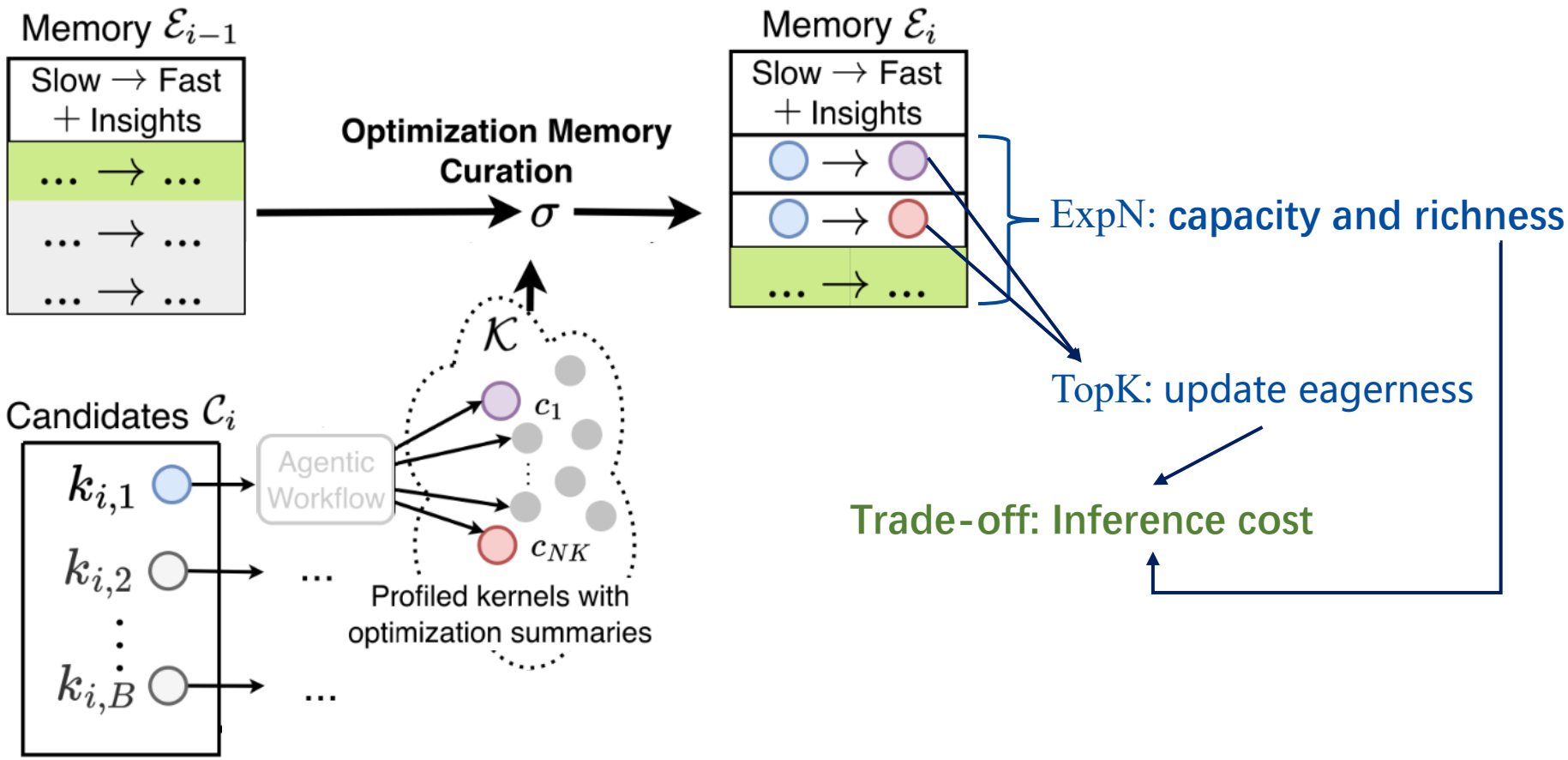
❑Capture optimization experience during exploration

Reserve latest experience



Design: Optmization Memory Curation

- Capture optimization experience during exploration



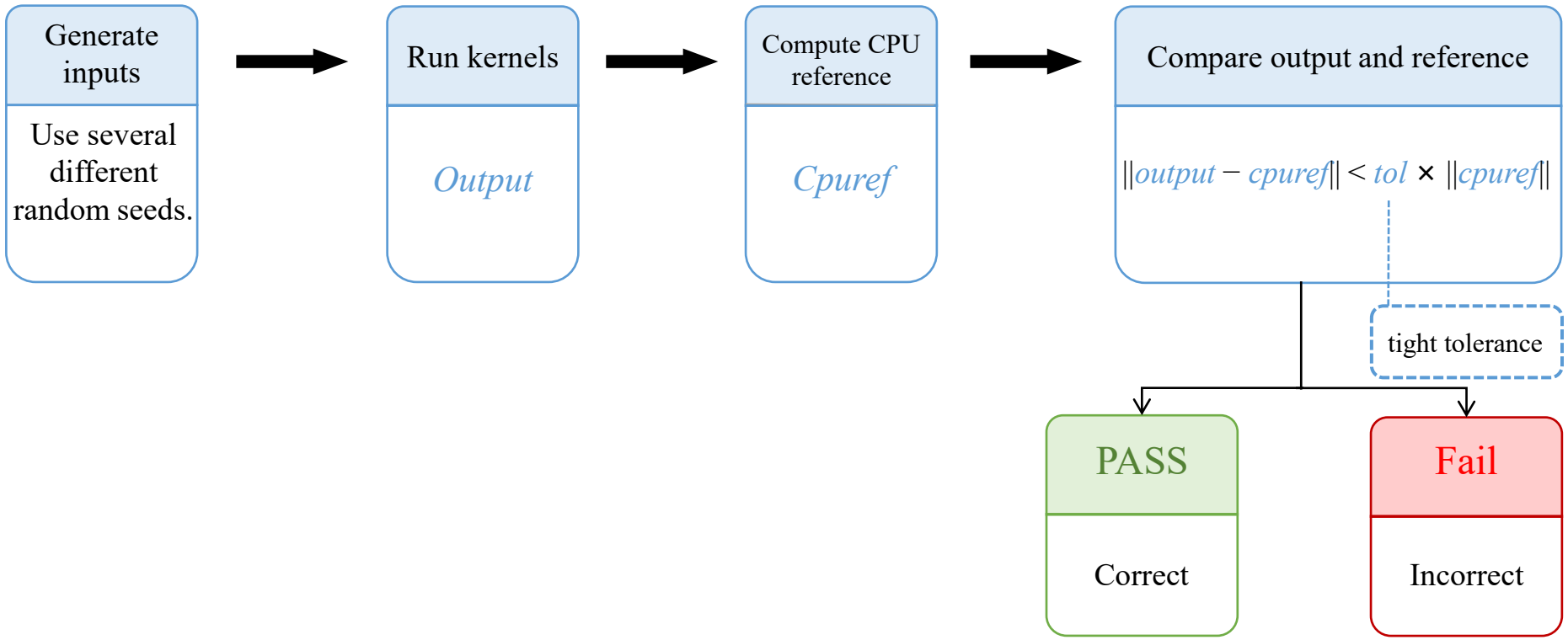
Profiling service

❑NKIBench Task Construction

- ❖14 representative NKI kernels from popular LLM workloads.
- ❖Created 4 NKI kernels
  - ➢optimized using standard techniques (tiling, fusion)
  - ➢adapted kernels from official NKI examples.
- ❖The benchmark includes inference and training kernels
  - ➢single operators (like Matmul andBatchMatmul)
  - ➢multi-operator chains (like Matmul+othersand LoRA)
  - ➢larger building blocks (like Group QueryAttention and Mamba block)

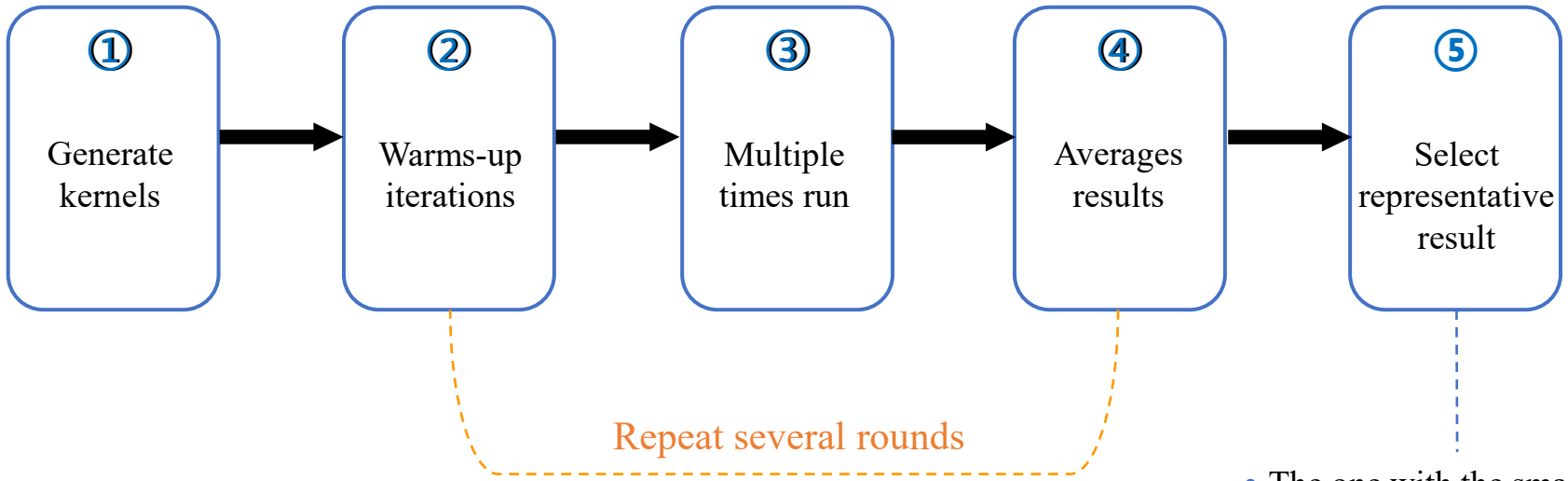
Profiling service

❑Correctness validation



Profiling service

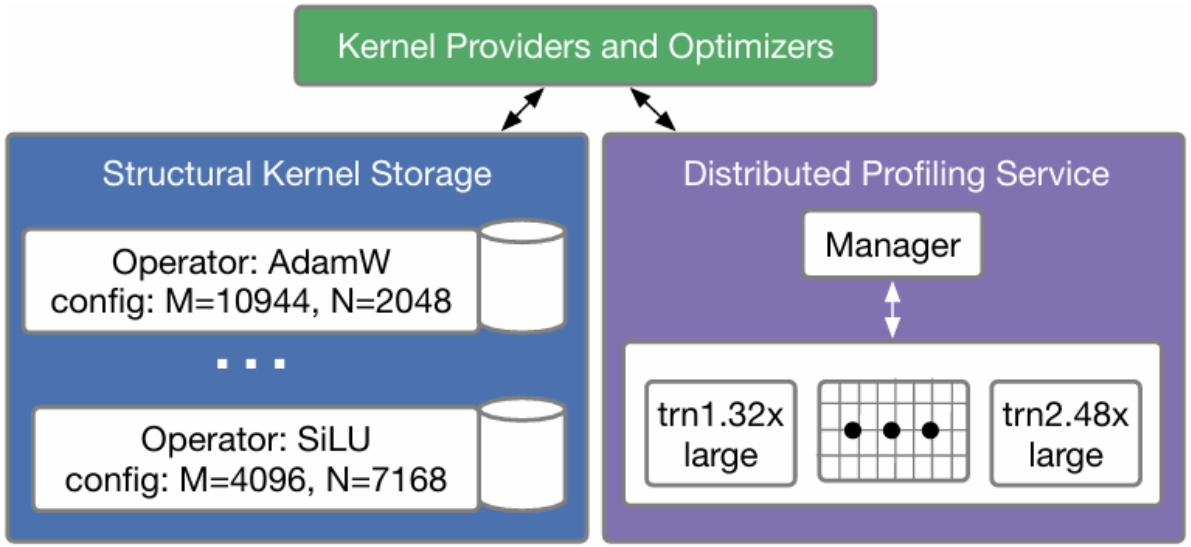
❑Performance result collection



Profiling service

❑Parallel profiling

- ❖Task-level: Each problem instance runs independently.
- ❖Sample-level: B × N × K kernels can be profiled simultaneously.



Evaluation: Setup

- **Claude Sonnet 4**
  - Using repeated sampling by querying the same prompt multiple times.
- **AccelOpt**
  - Planner and Summarizer: gpt-oss-120b.
  - Executor: Qwen3-Coder-480B-A35B-Instruct-FP8.
- **Other Parameters**
  - $t_{pos} = 1.04$, $t_{neg} = 1.04$.
  - $\mathrm{TopK} = 8, \mathrm{ExpN} = 16, \mathrm{B} = 6, \mathrm{N} = 12, \text{and } \mathrm{T} = 16.$

Evaluation: Setup

❑Base Prompt for NKI Kernel Optimization

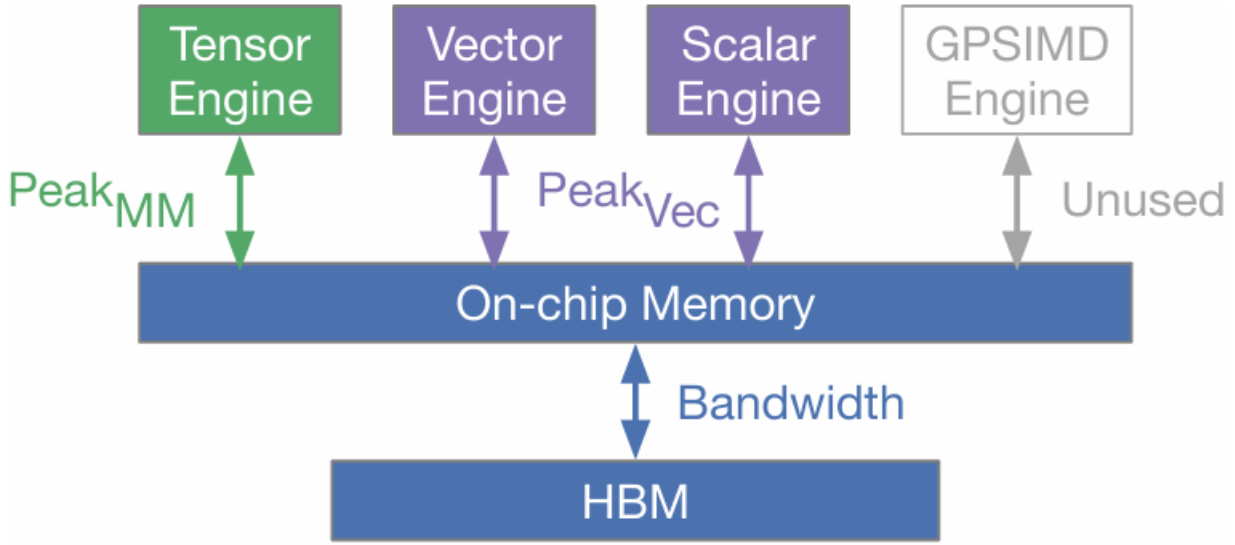
- ❖Role: Set LLM as NKI kernel optimization expert.
- ❖Input: Problem code, Baseline kernel, Kernel usage, Profile.
- ❖Goal: Reduce kernel latency while preserving correctness
- ❖Workflow:
  - ➢Analyze the profile.
  - ➢Identify bottlenecks.
  - ➢Generate an optimization plan.
  - ➢Output the optimized kernel.

Evaluation: Setup

- **Knowledge Injection and Correctness Constraints**
  - Optimization knowledge.
    - tiling, fusion, spill reduction...
  - NKI hardware constraints.
  - Profile feedback.
    - memory traffic, spill behavior, hardware utilization, and kernel latency.
  - Code rules:
    - indexing rules, tensor scope...

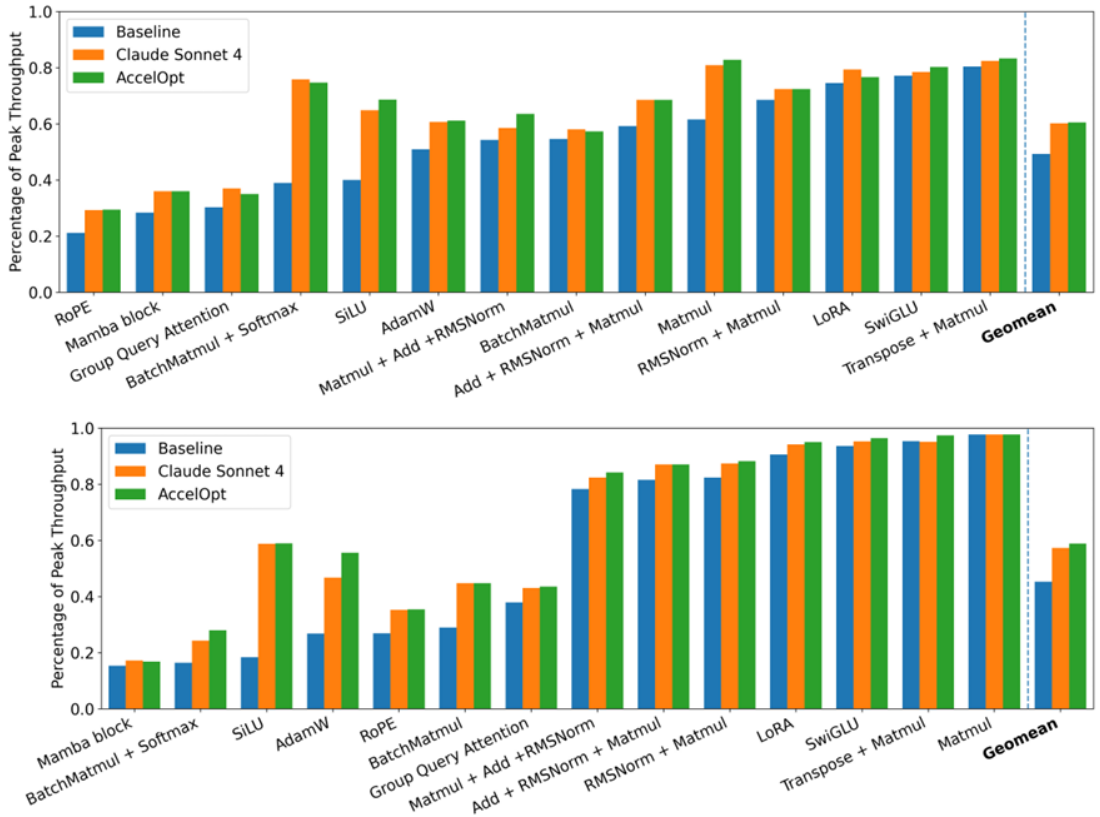
Peak Performance Calculation

- **Tensor, vector, and scalar engines run concurrently.**
  - Peak performance: $T = \max\left(\frac{\text{Traffic}_{\text{Min}}}{\text{Bandwidth}}, \frac{\text{FLOPs}_{\text{MM}}}{\text{Peak}_{\text{MM}}}, \frac{\text{FLOPs}_{\text{Vec}}}{\text{Peak}_{\text{Vec}}}\right)$
  - The percentage of peak throughput : $\frac{T}{t}$



Evaluation: Overall

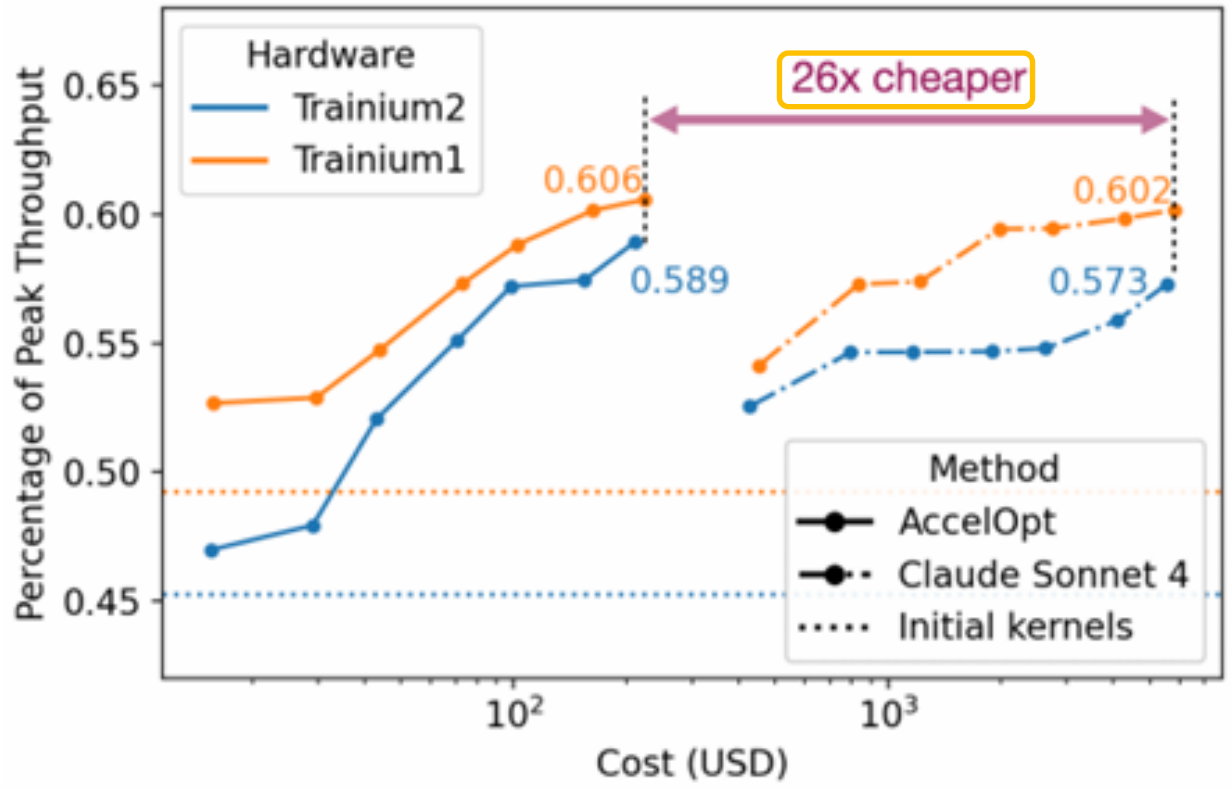
- **Overall:** Compare AccelOpt using open-source LLMs with repeated sampling of Claude Sonnet 4



Comparable performance!

Evaluation: Overall

- **Cost:** Compare AccelOpt using open-source LLMs with repeated sampling of Claude Sonnet 4



The average throughput:

	Before	After
Trainium 1	49%	61%
Trainium 2	45%	59%

Cost:

AccelOpt : Claude Sonnet 4 = 1 : 26

Evaluation: Optimization Case

❑Peephole Optimization:

- ❖ $\theta_{t-1} - \gamma\lambda\theta_{t-1} \Rightarrow (1 - \gamma\lambda)\theta_{t-1}$
- ❖ $\text{reciprocal}(\text{sqrt}(\ldots))) \Rightarrow \text{rsqrt}(\ldots))$
- ❖ $x/(1 + e^{-x}) \Rightarrow x \cdot \text{sigmoid}(x)$

Local Optimization

❑Educational Impact:

- ❖In Stanford CS149 Fall 2025, 33.6% of 131 teams of students conquered an extra credit problem **based on AccelOpt**.

Stanford CS149, Fall 2025

PARALLEL COMPUTING

From smart phones, to multi-core CPUs, to GPUs, to AI accelerators, to the world's largest supercomputers and web sites, parallel processing is ubiquitous in modern computing. The goal of this course is to provide a deep understanding of the fundamental principles and engineering trade-offs involved in designing modern parallel computing systems as well as to teach parallel programming techniques necessary to effectively utilize these machines. Because writing good parallel programs requires an understanding of key machine performance characteristics, this course will cover both parallel hardware and software design.

Assignment 4: Programming a Machine Learning Accelerator

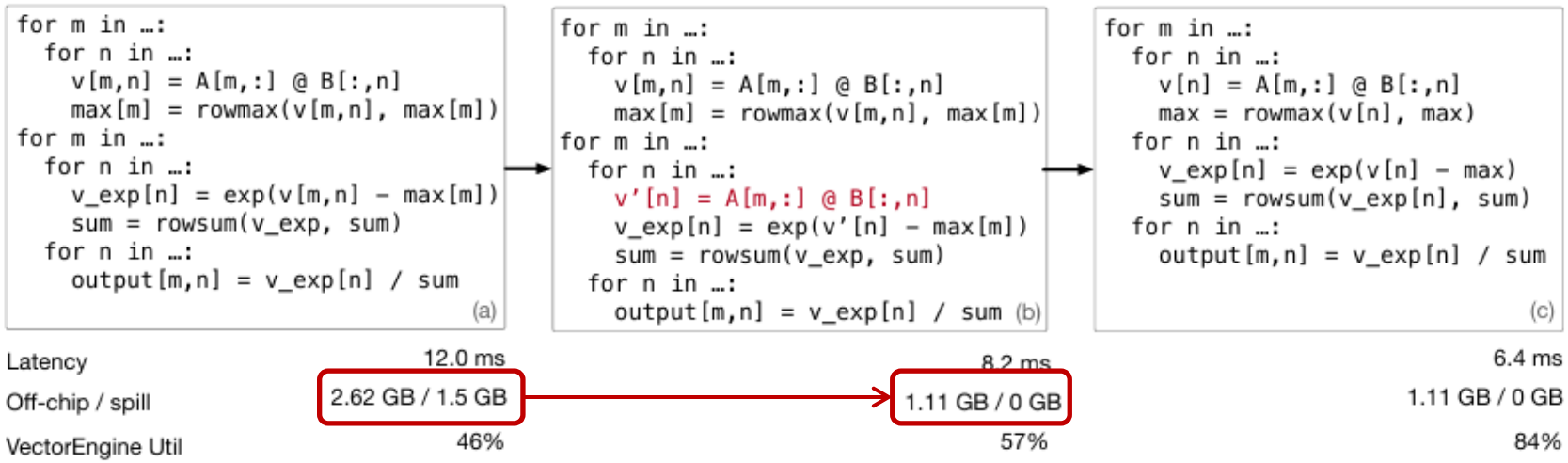
Due Thursday Nov 13, 11:59pm

100 points total

Evaluation: Optimization Case

❑Loop Optimization:

❖AccelOpt can also discover non-local optimizations.



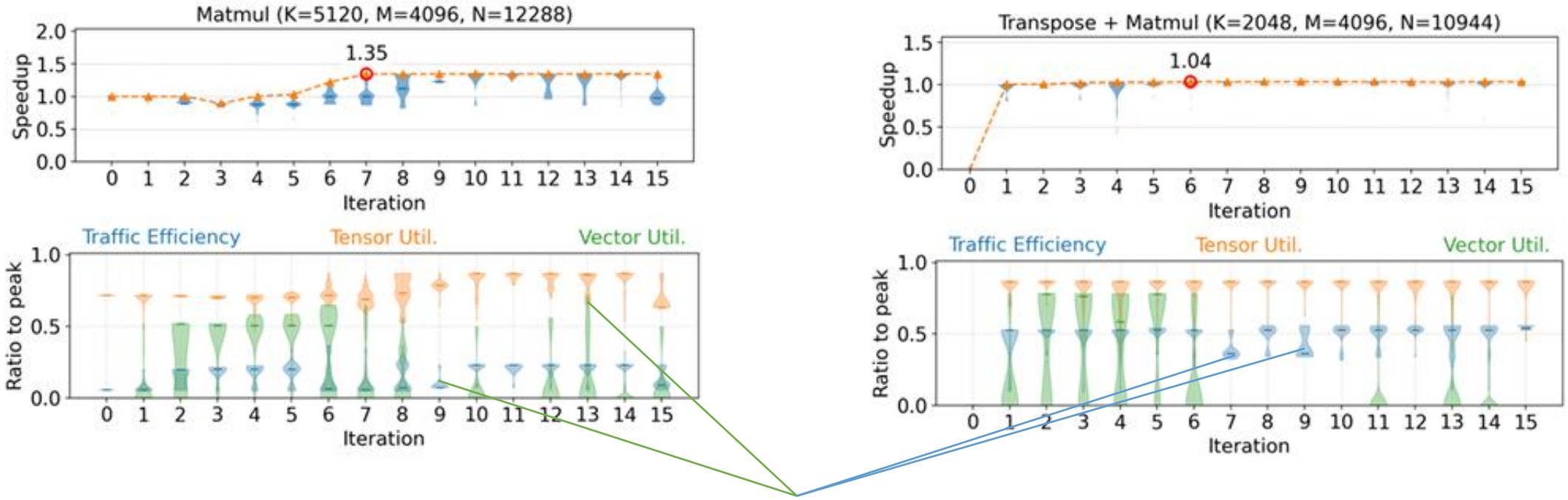
Off-chip Spill ✗
Low Performance ✗

No Spill ✓
Extra Recomputation ✗

No Spill ✓
High Performance ✓

Evaluation: Optimization Limitation

- **AccelOpt cannot further optimize the kernels**
  - The kernel is close to peak.

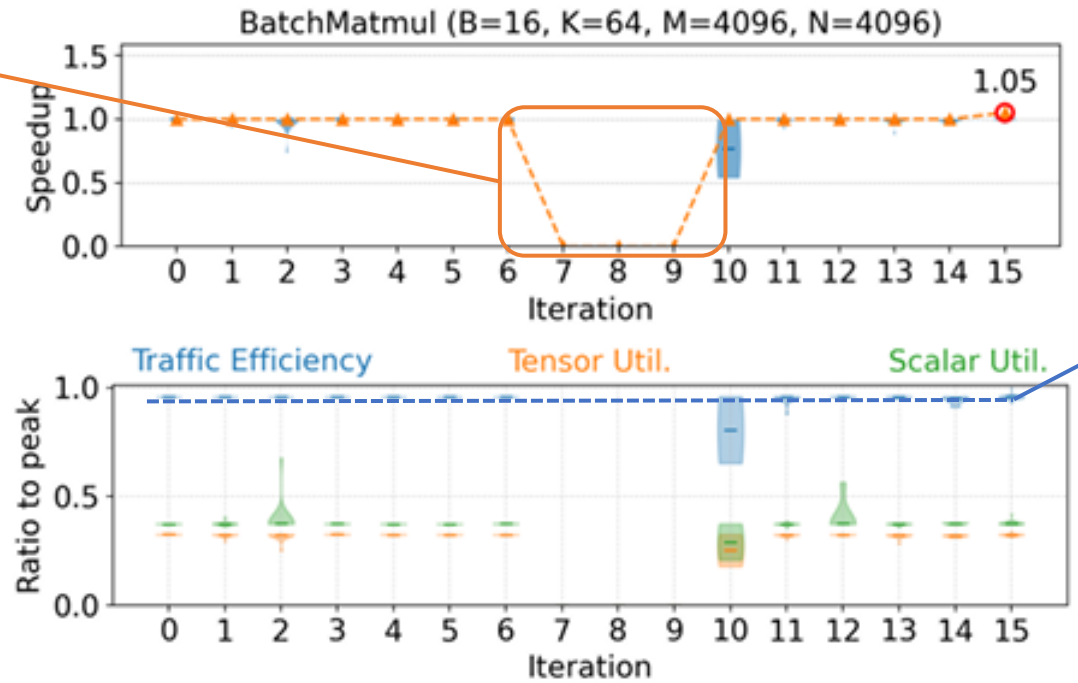


Exploration mechanism remains active

Evaluation: Optimization Limitation

- **AccelOpt cannot further optimize the kernels**
  - The initial kernel is challenging to optimize.

No correct kernels

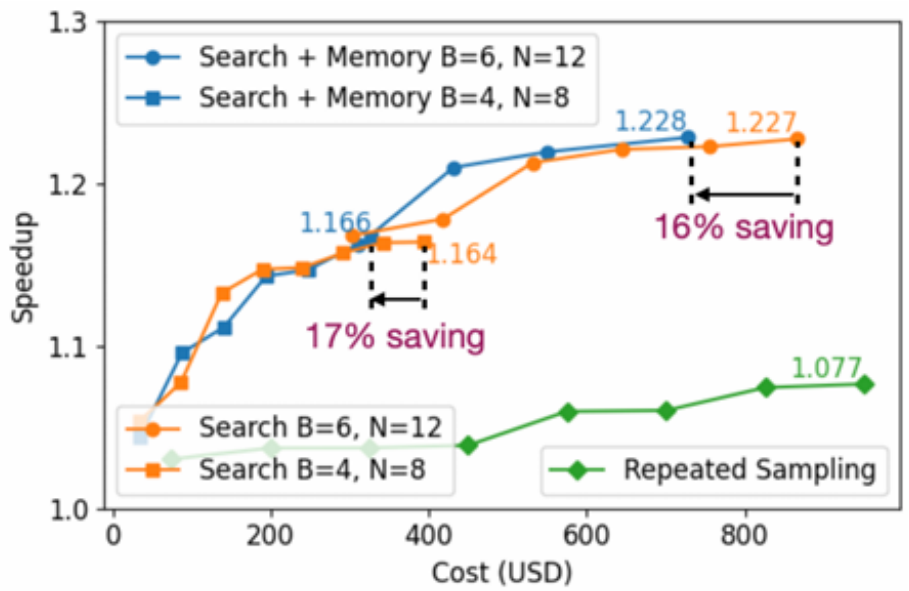


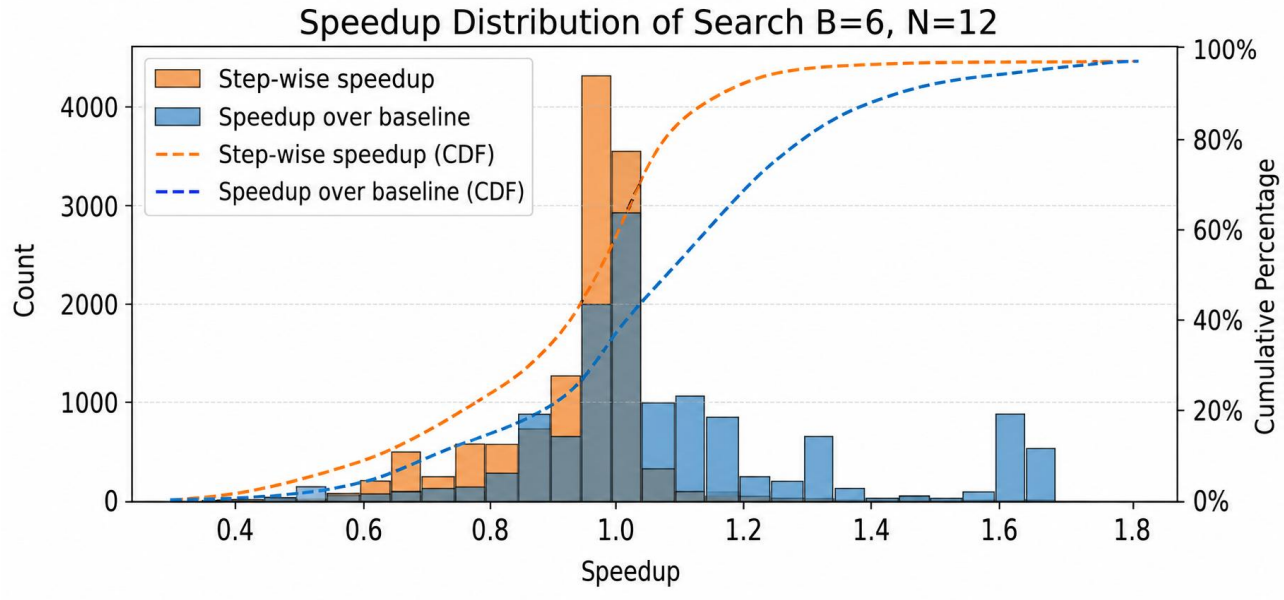
Traffic Efficiency Near 1.0

Evaluation: Ablation Study

❑Beam Search vs. Repeated Sampling Only

❖Each iteration builds upon previous best kernels.





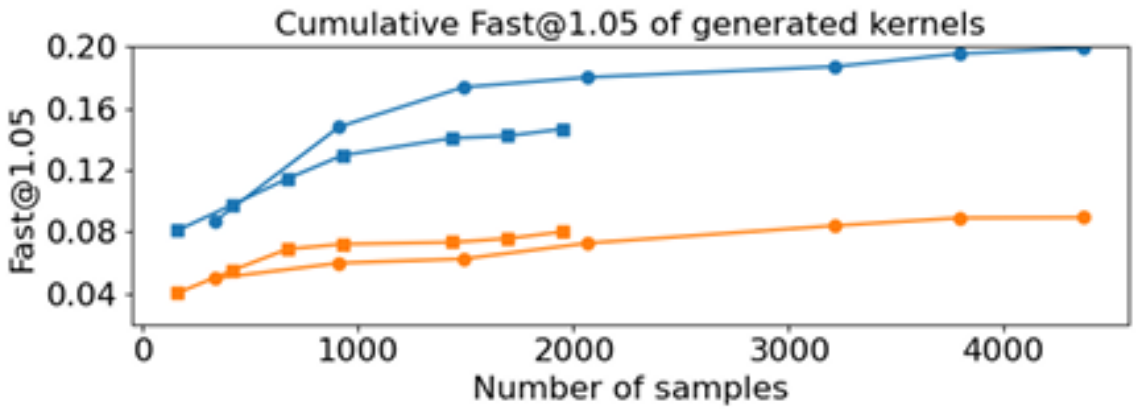
Beam Search → **Higher speedup with lower cost**

Evaluation: Ablation Study

❑Optimization Memory vs. BeamSearchOnly

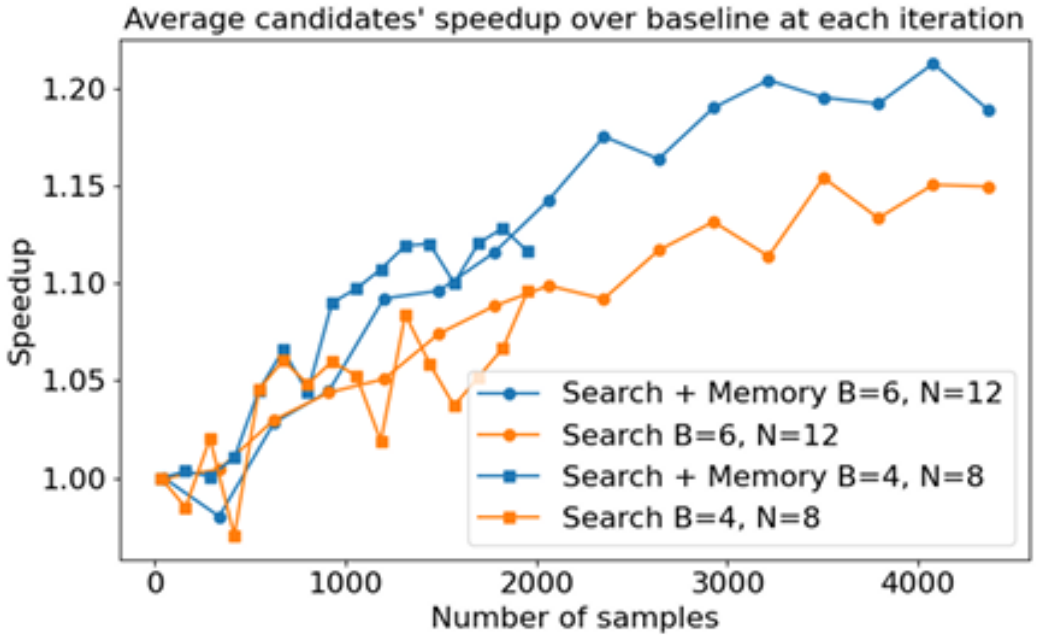
❖Optimization memory increases the probability of generating fast kernels

7 rounds



Fast@1.05: fraction of generated kernels with speedup > 1.05

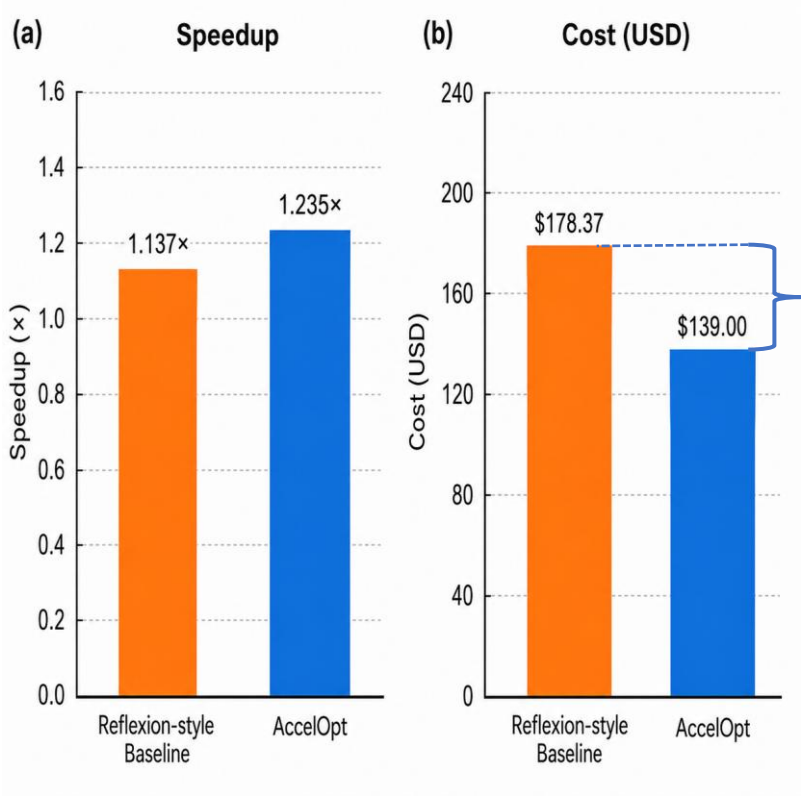
16 rounds



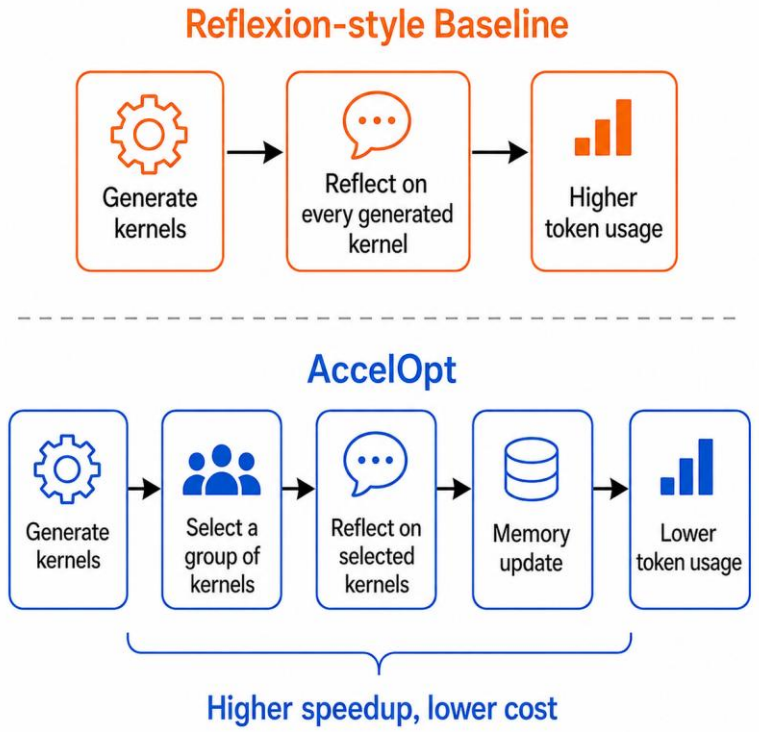
Evaluation: Ablation Study

❑Reflexion-style vs. AccelOpt

- ❖Reflexion-style: reflects on **every generated kernel**.
- ❖AccelOpt: reflects on **a selected group of kernels**.



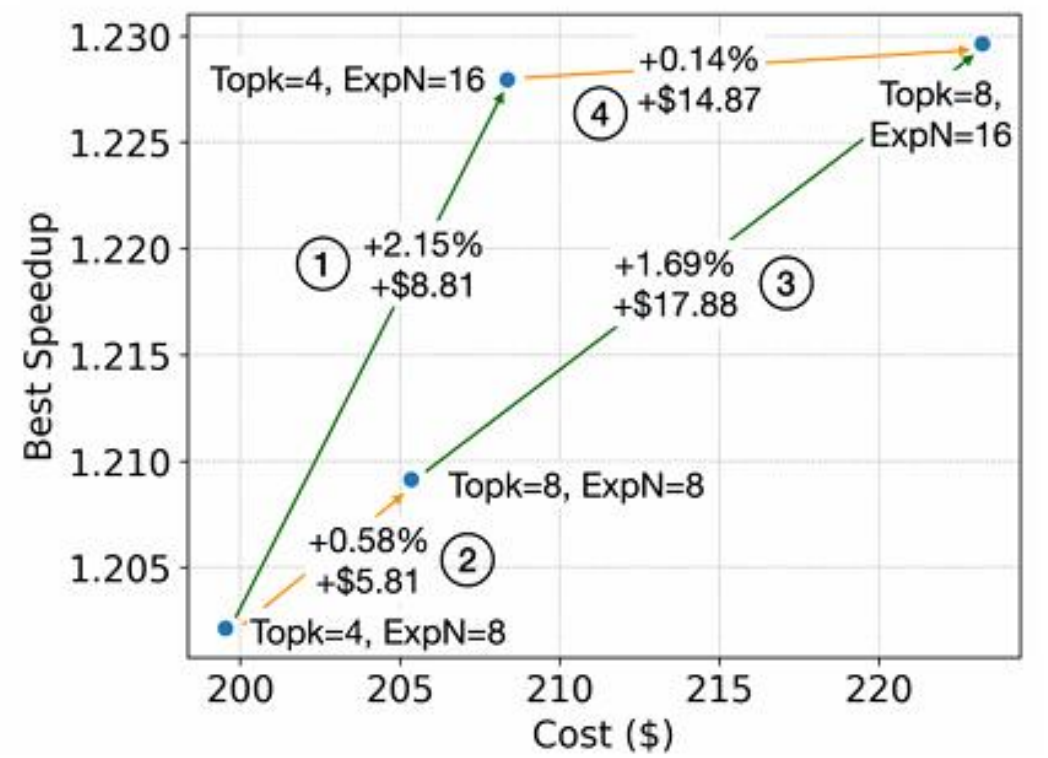
$39.37!



Evaluation: CostAnalysis

❑Capacity (ExpN) vs. Update eagerness (TopK)

- ❖The delta of speedup is much larger when increasing ExpN.①③
- ❖The effectiveness of increasing ExpN depends on the model.



Executor \ ExpN	8	16	Delta	
Qwen3-Coder-30B-A3B-Instruct	1.144 $96.10	1.197 $108.43	+4.6% +$12.33	**Cost-effective**
gpt-oss-120b	1.228 $125.19	1.235 $139.00	+0.6% +$13.81	**Best performance**
Qwen3-Coder-480B-A35B-Instruct-FP8	1.209 $205.35	1.230 $223.23	+1.7% +$17.88	
Ensemble (Best of three models)	1.241 $426.64	**1.246** $470.66	+0.4% +$44.02	

Performance depends on the executor.

Evaluation: CostAnalysis

❑Base Model Selection: Executor Matters More

Switching Planner: Little Speedup Difference

Planner	Speedup	Cost
gpt-oss-20b	1.234	$116.87
gpt-oss-120b	1.235	$139.00
Qwen3-235B-Thinking	1.234	$316.21

Claude as Backbone: Much Higher Cost

Planner	Executor	Speedup (Cost)
Claude Sonnet 4	Claude Sonnet 4	1.226 ($1732.73)
gpt-oss-120b	Claude Sonnet 4	1.213 ($1269.98)
Claude Sonnet 4	gpt-oss-120b	1.208 ($1223.05)
gpt-oss-120b	gpt-oss-120b	1.235 ($139.00)

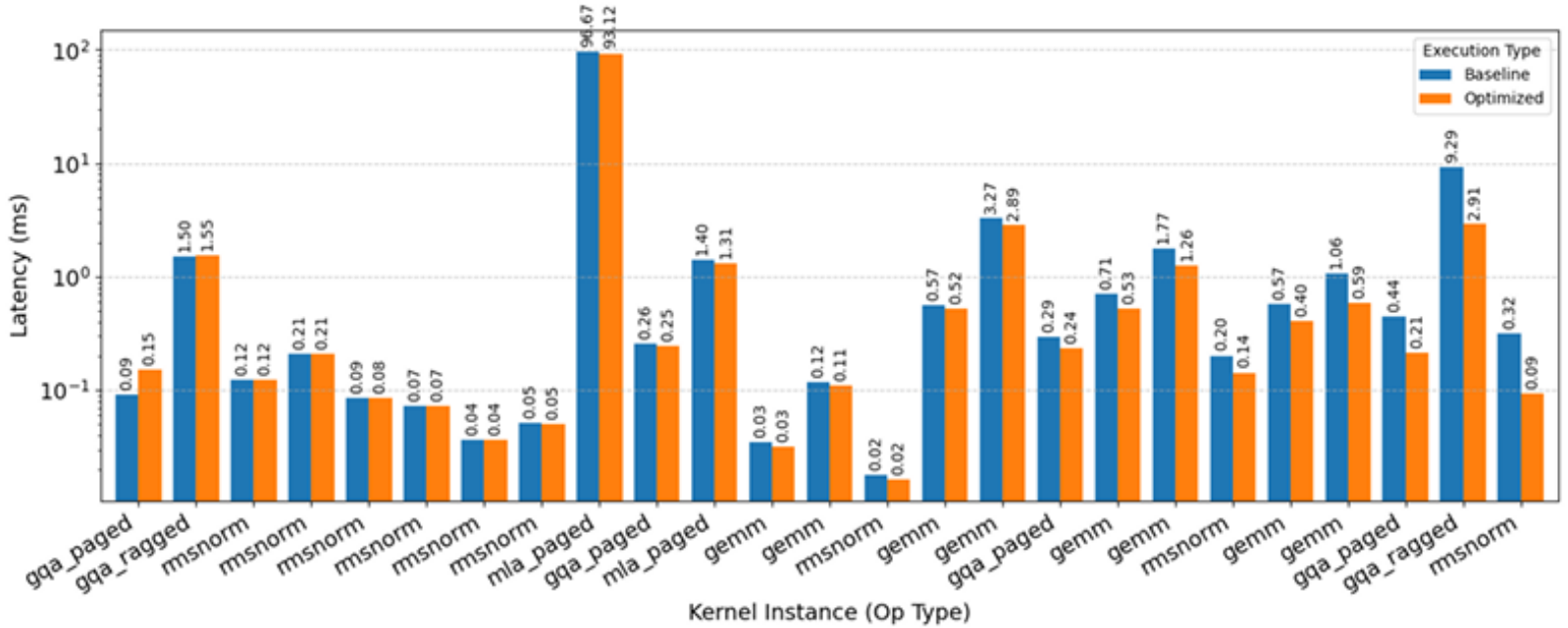
❑Conclusion

- ❖Planner basically does not affect speedup, but affect cost.
- ❖gpt-oss-120b achieves the best cost-performance.

Conclusion

❑AccelOpt is platform-agnostic

- ❖Adaptation requires:
  - ➢a profiling service.
  - ➢platform-specific base prompts.



AccelOpt can optimize Triton kernels of FlashInfer-Bench

Conclusion

- **AccelOpt is the first self-improving LLM agentic system for AI accelerator kernel optimization.**
  - Beam search + optimization memory improve kernel performance and cost efficiency.
  - Requires no expert-provided optimization knowledge.

AccelOpt and NKIBench provide a promising foundation for automated kernel optimization on emerging AI accelerators.