

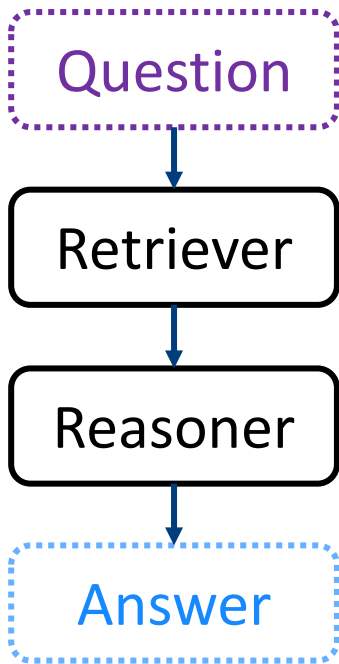
SCOPE: Cost-Efficient Model Selection for Compound AI Systems under Quality Constraints

Yiqian Huang¹, Shiqi Zhang¹, Tianyuan Jin², Xiaokui Xiao¹

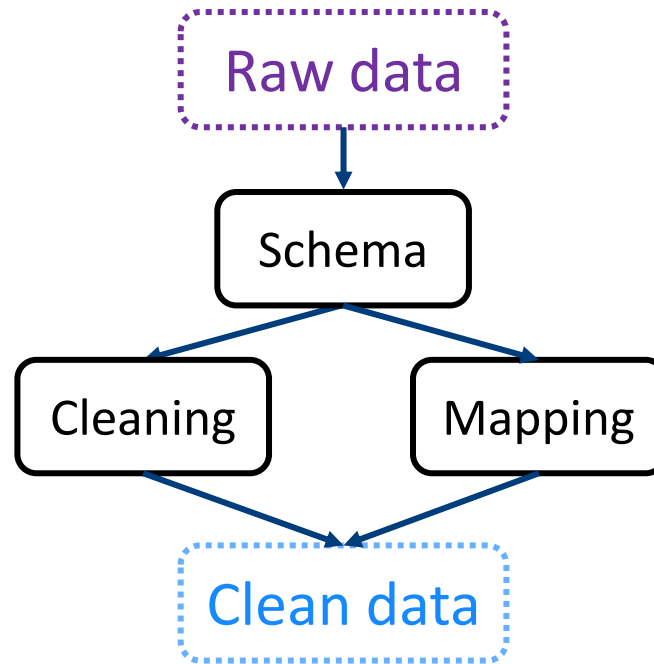


Compound AI Systems

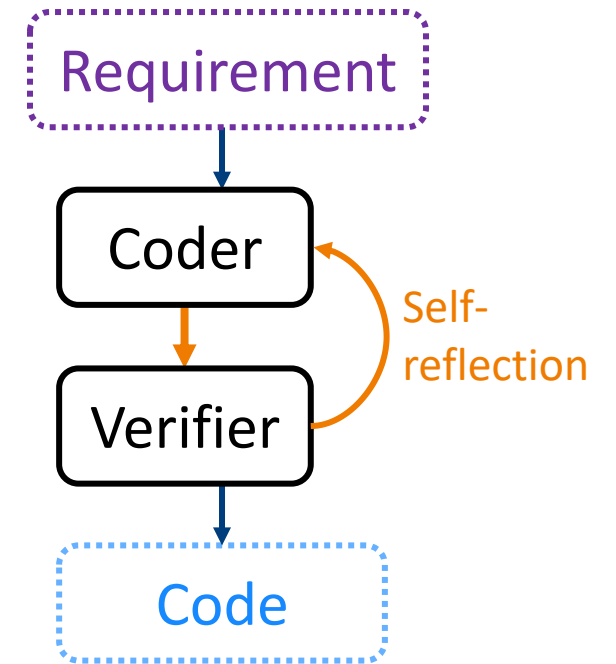
- A compound AI system is a system made up of multiple AI agents



Question answering



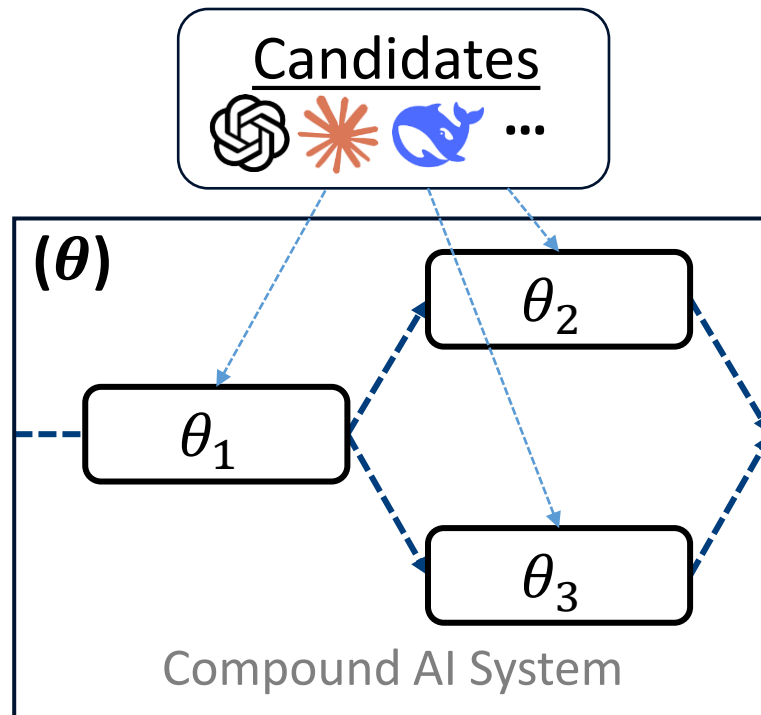
Data processing



Code development

Model Selection

- To run the system, we should decide which models to use

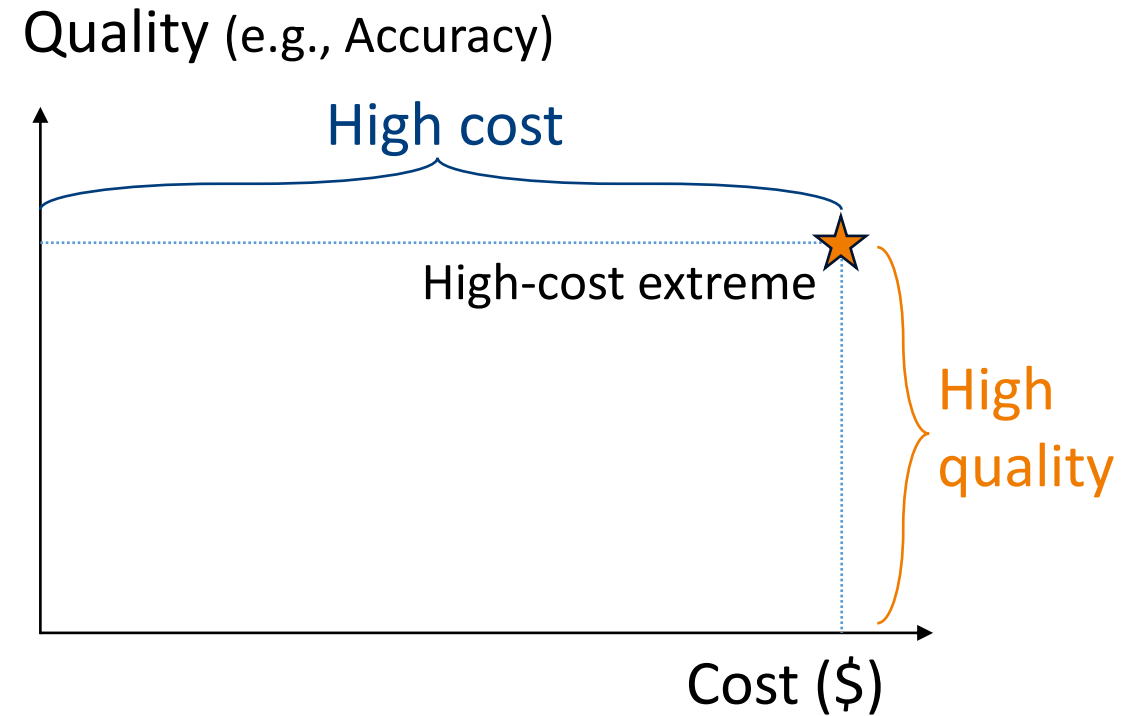
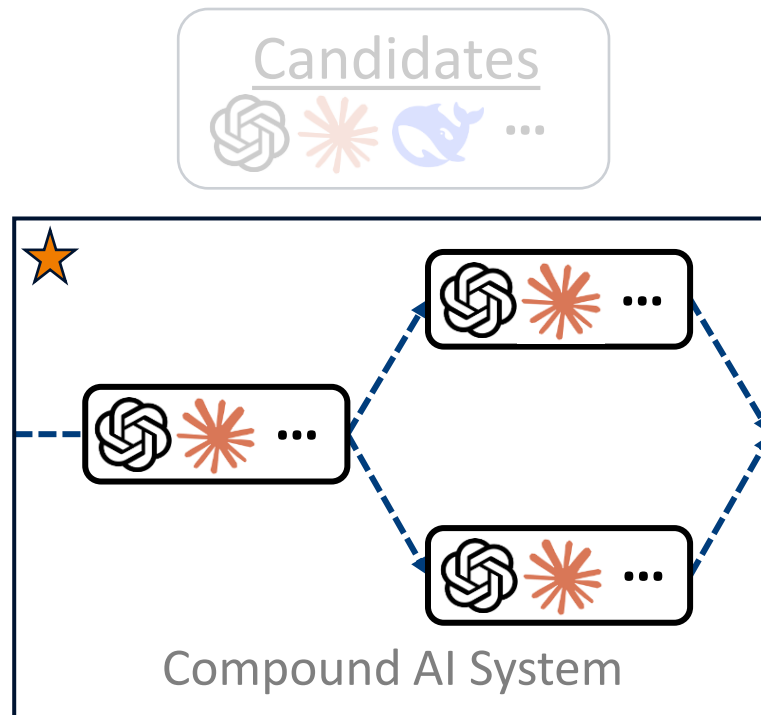


Quality (e.g., Accuracy)

Cost (\$)

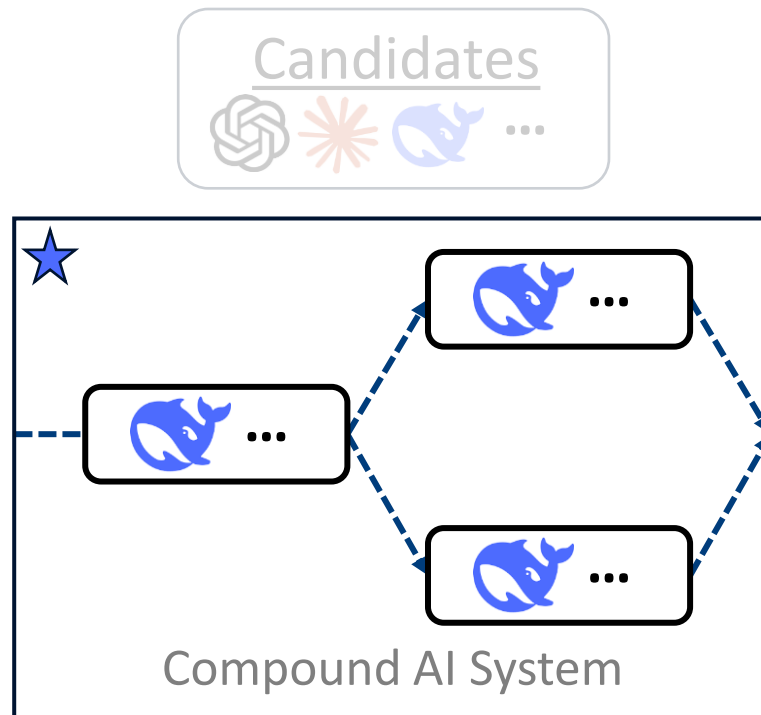
Model Selection

- We may use **High-cost extreme** ★ (the most expensive models)

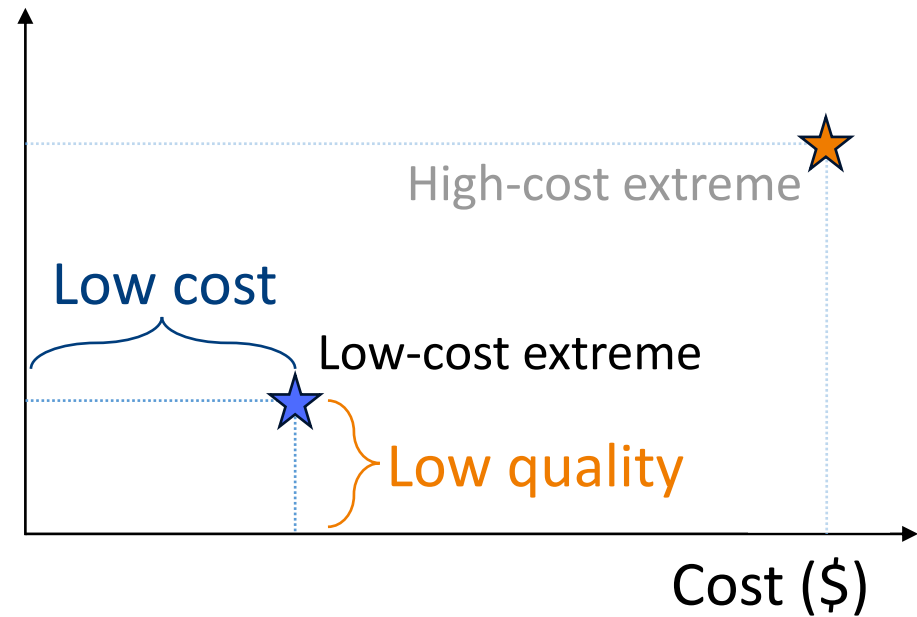


Model Selection

- Or we may use **Low-cost extreme** ★ (the most affordable models)

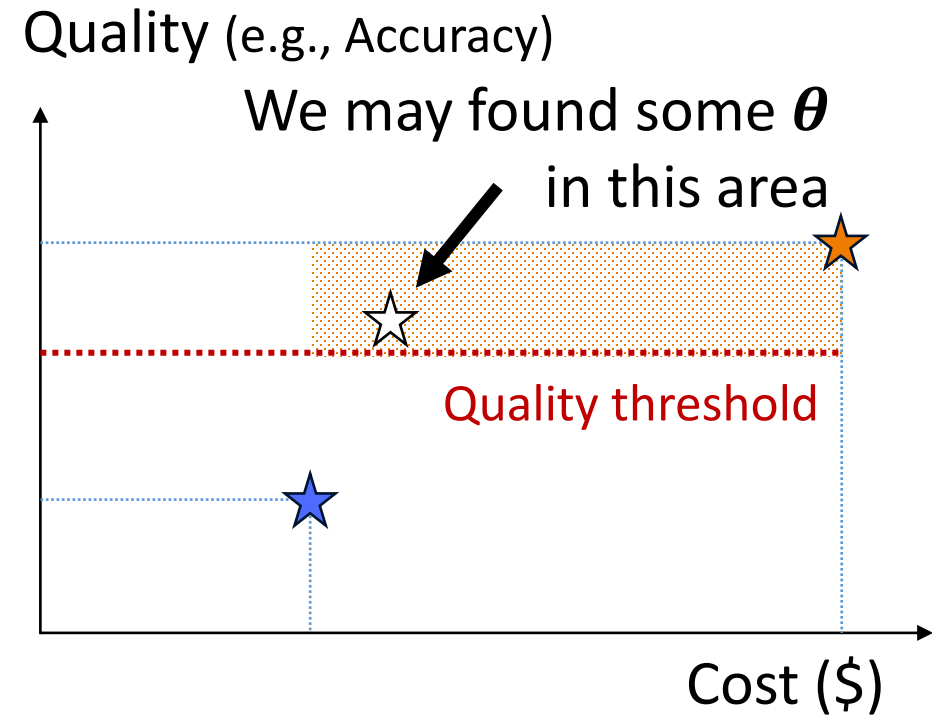
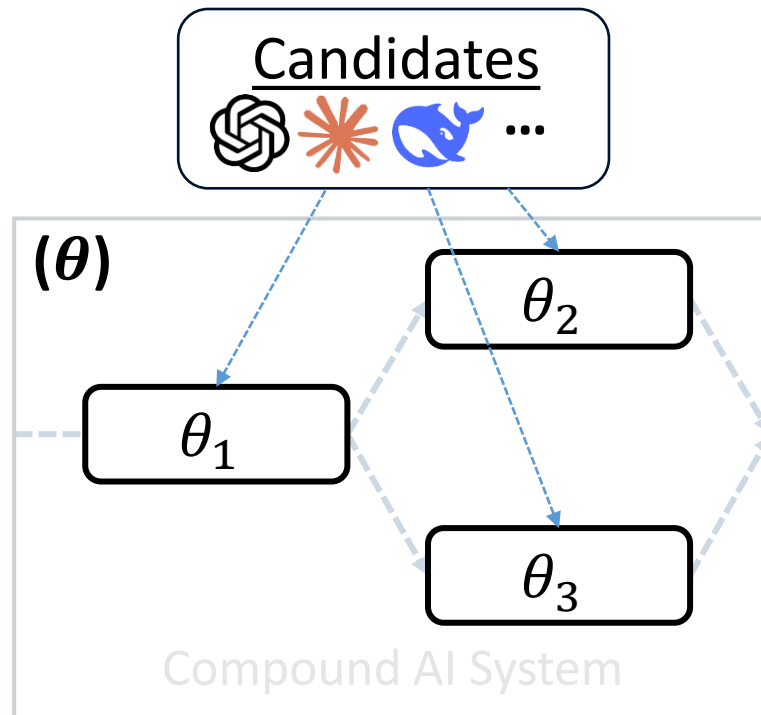


Quality (e.g., Accuracy)



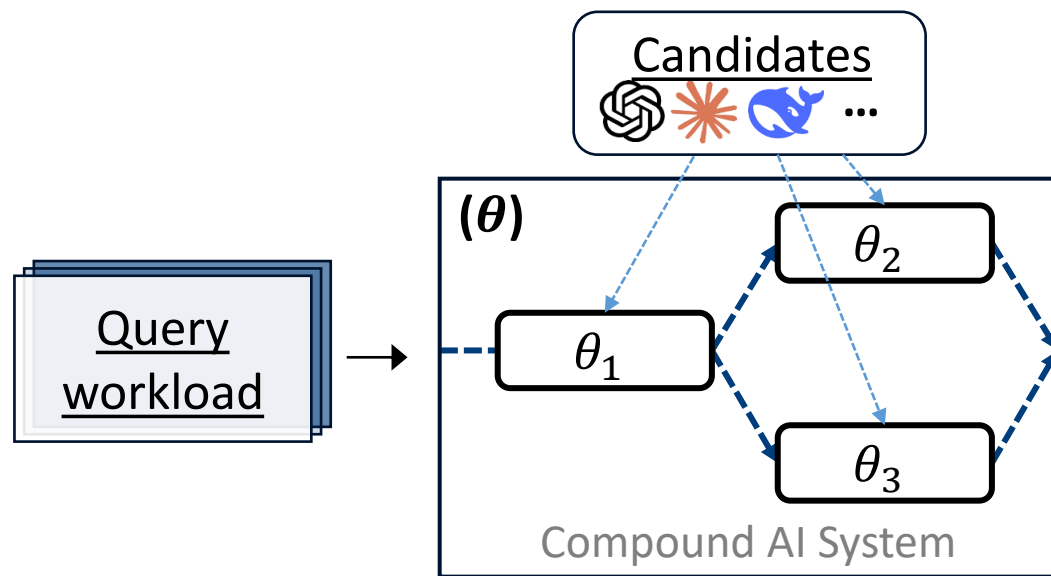
Model Selection

- Can we *preserve most of the quality* from ★ while saving cost?



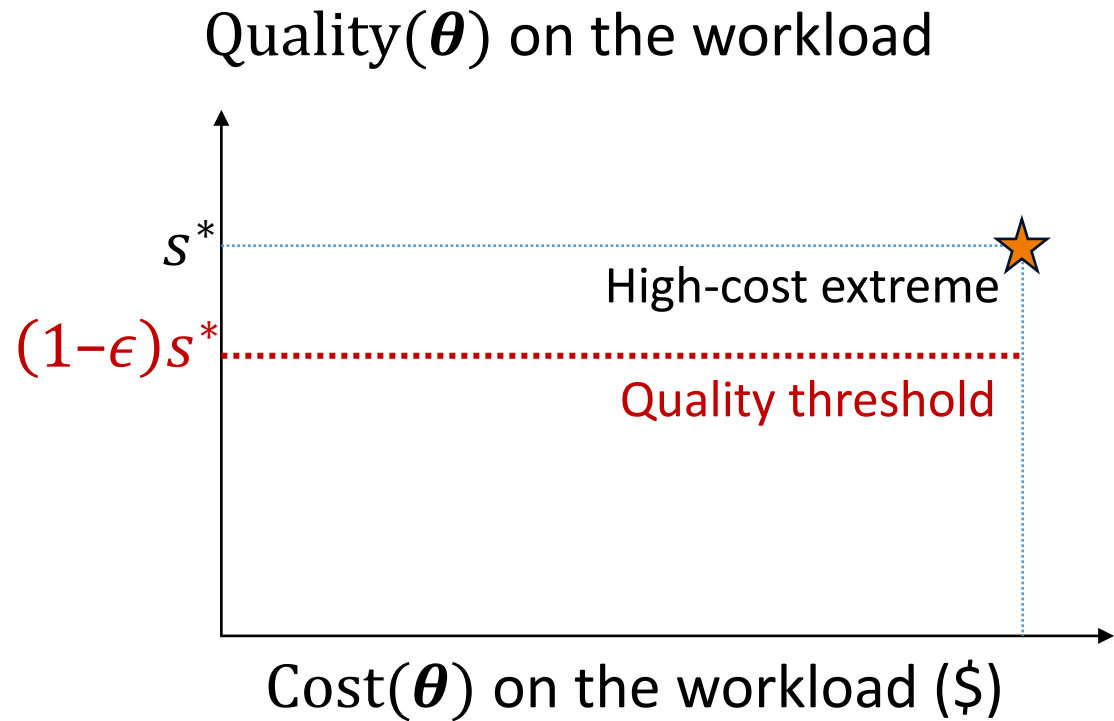
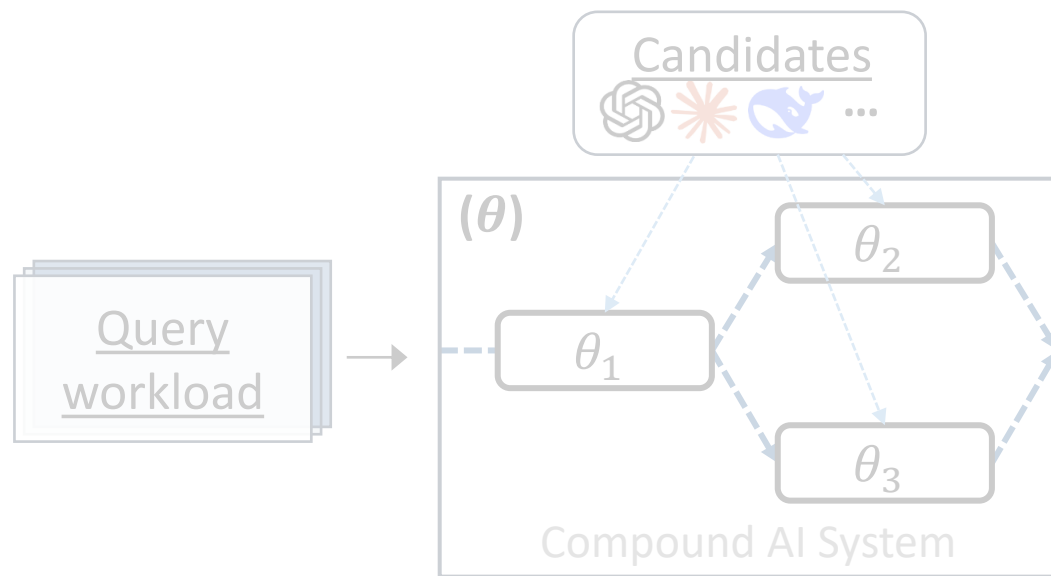
Problem Definition

- **Problem setup:** Compound AI system + Query workload + Candidate LLMs



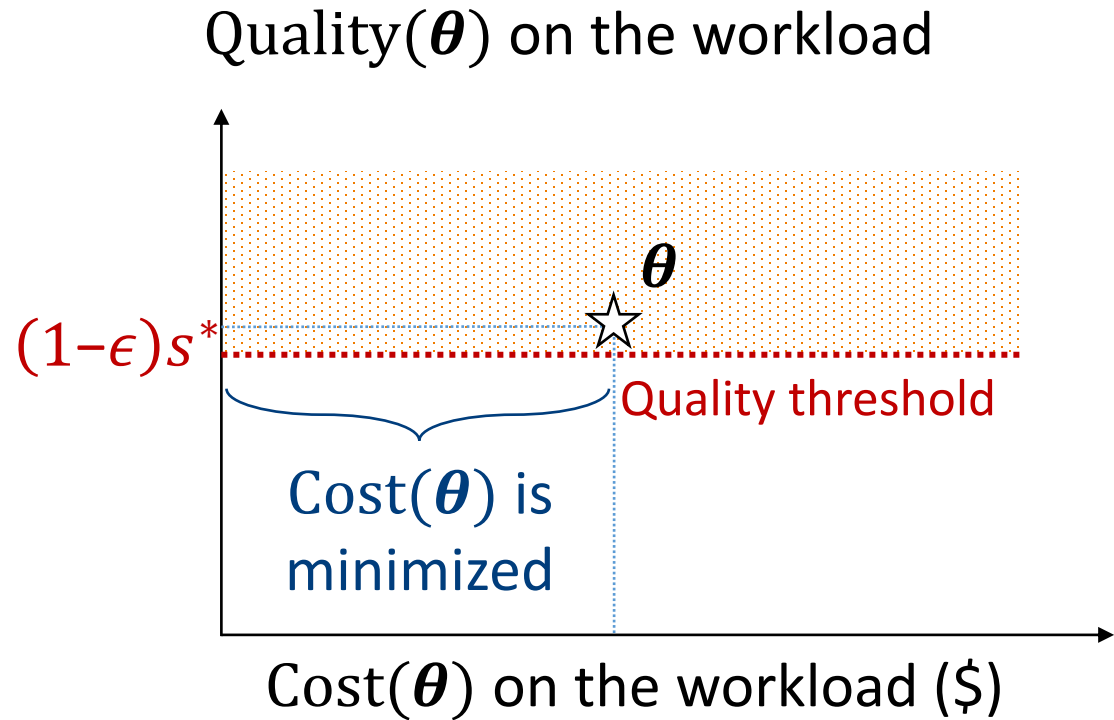
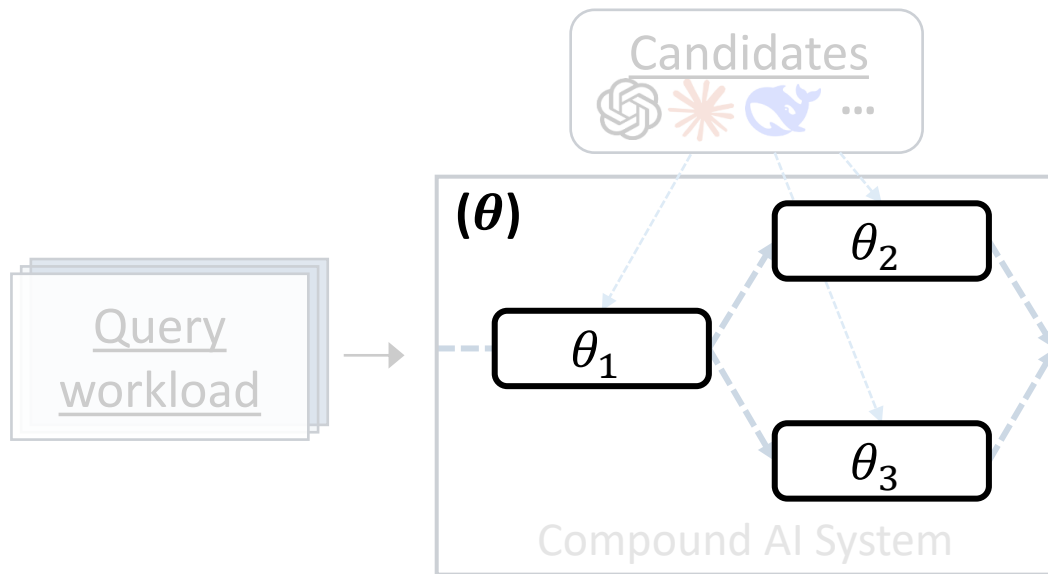
Problem Definition

- **Problem setup:** Compound AI system + Query workload + Candidate LLMs
- **Input:** s^* (quality of high-cost extreme), ϵ (allowable quality degradation)



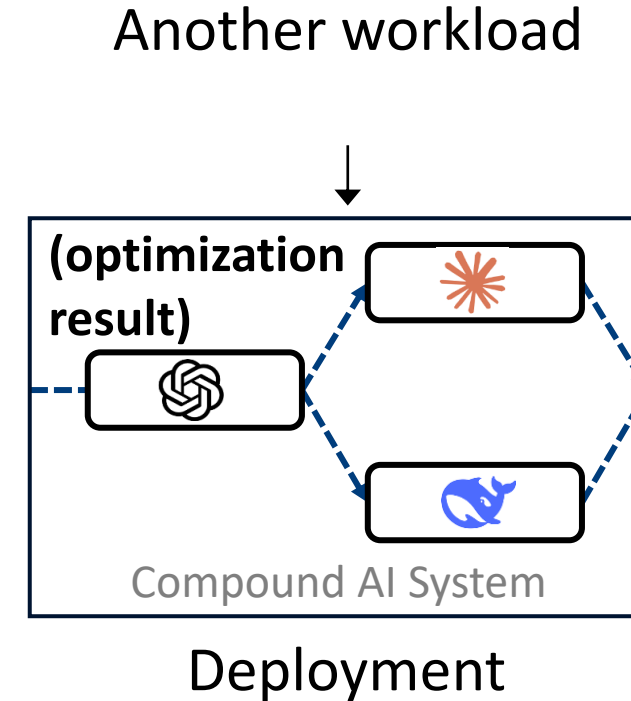
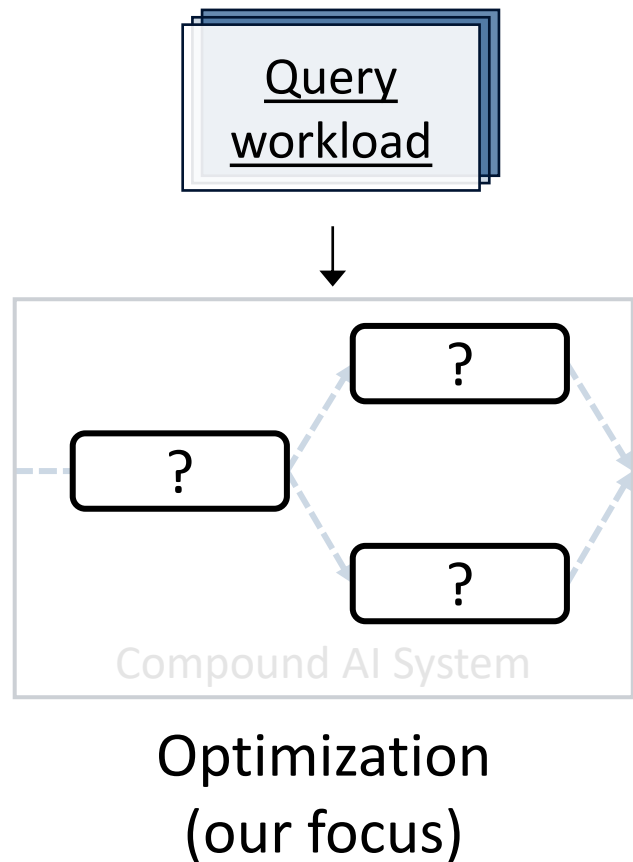
Problem Definition

- **Objective:** Minimize $\text{Cost}(\theta)$ subject to $\text{Quality}(\theta) \geq (1-\epsilon)s^*$



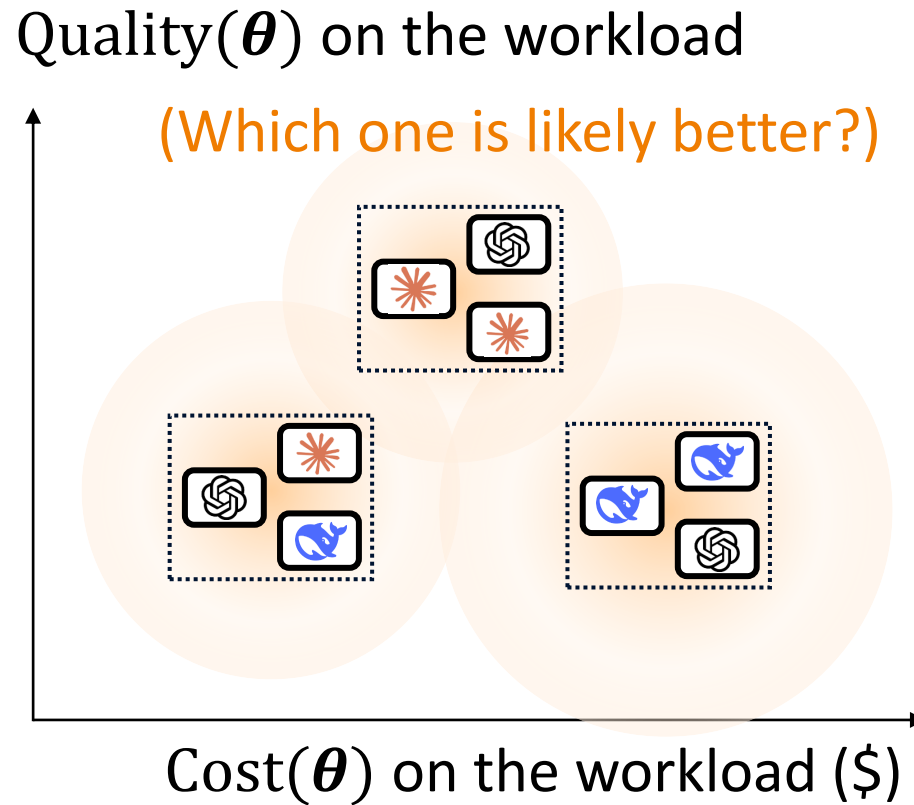
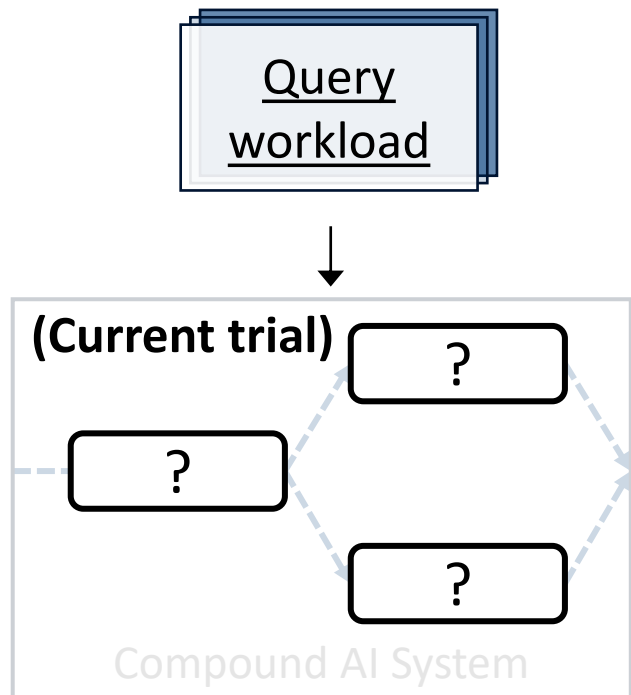
Problem Definition

- We use the given workload only to optimize configurations



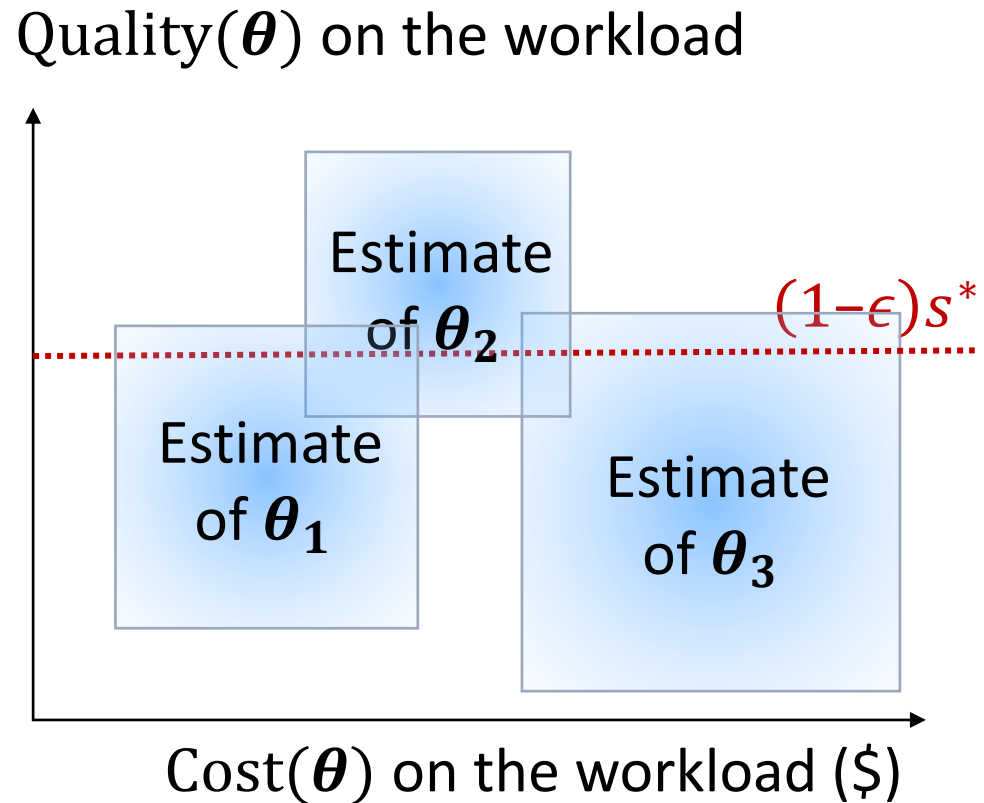
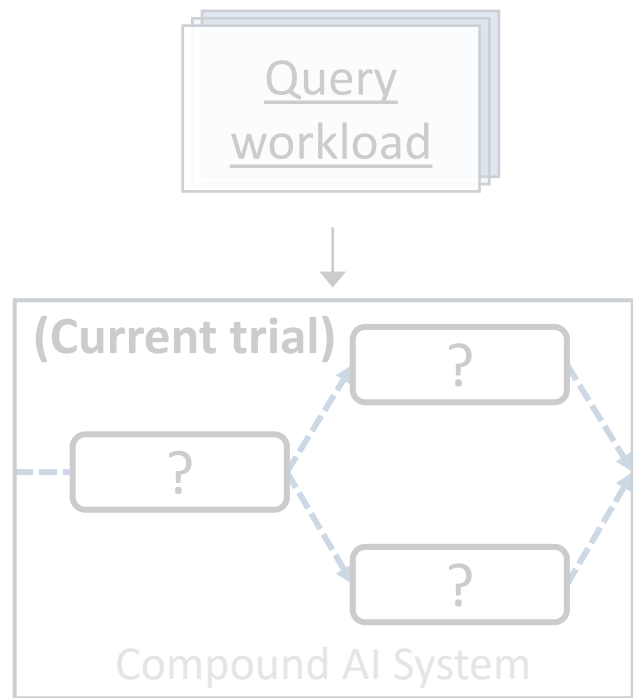
Existing Solutions

- Existing solutions use *the whole workload* to choose the next configuration



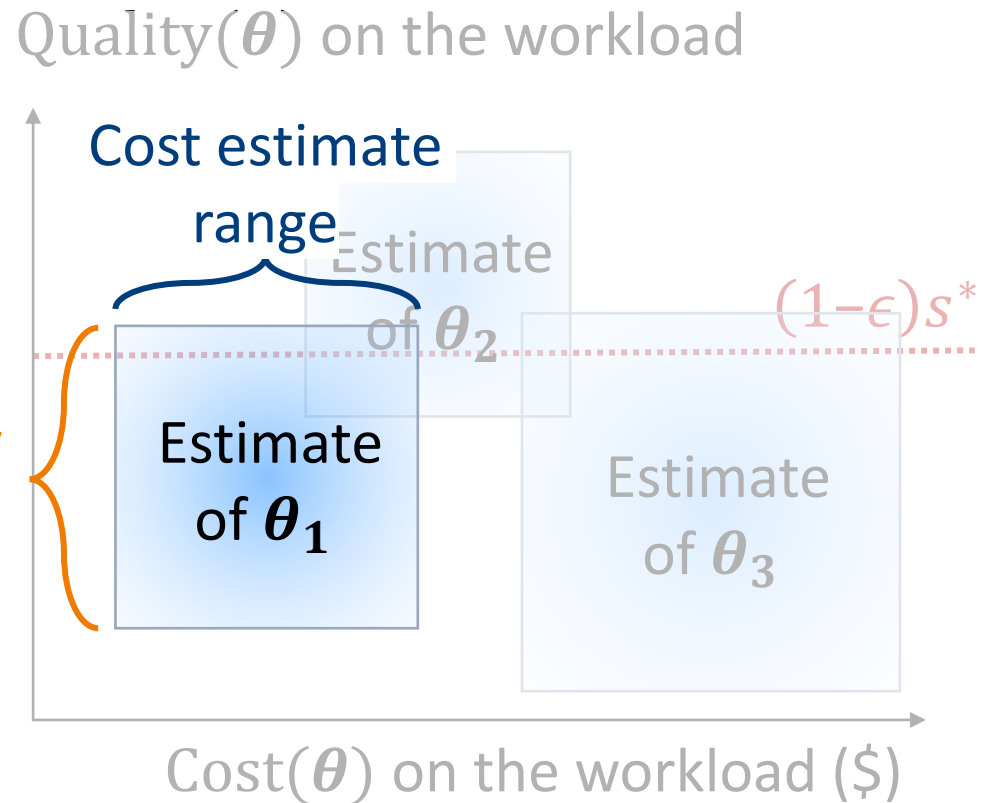
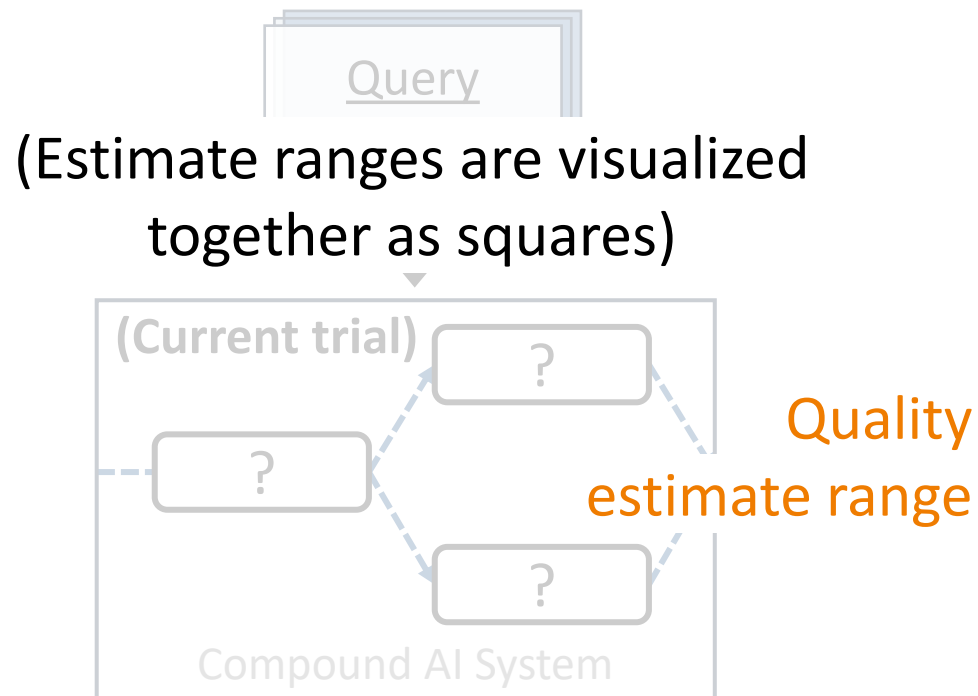
Existing Solutions (Bandit as an example)

- Example: Bandit methods keep estimates for configurations



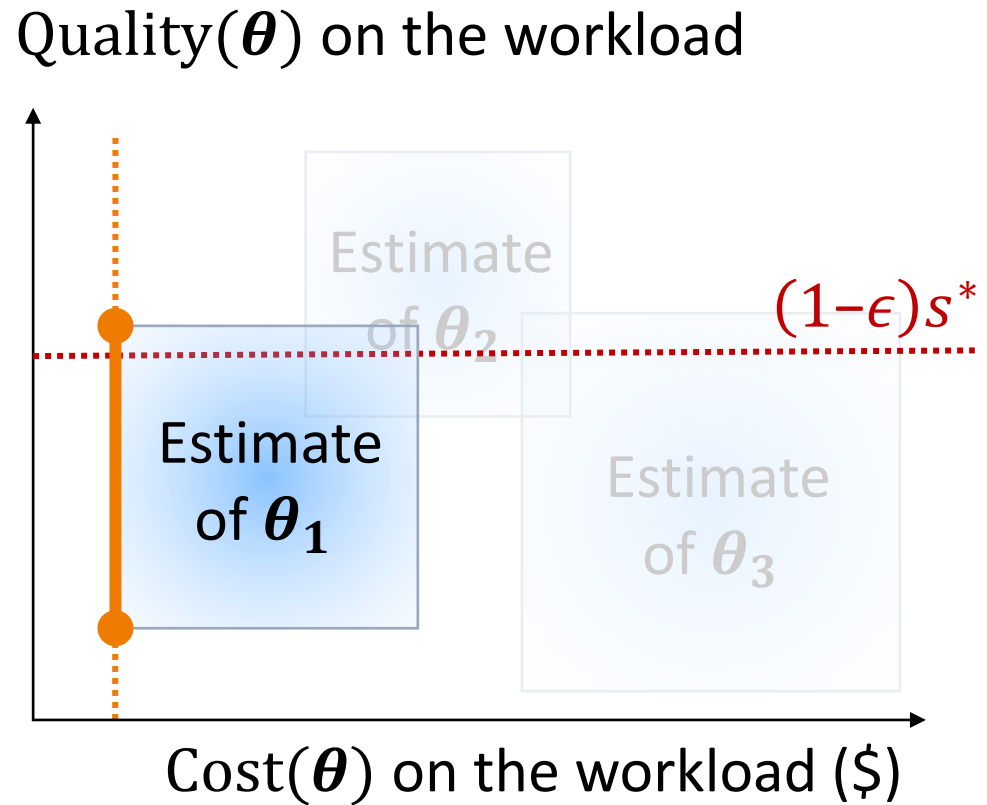
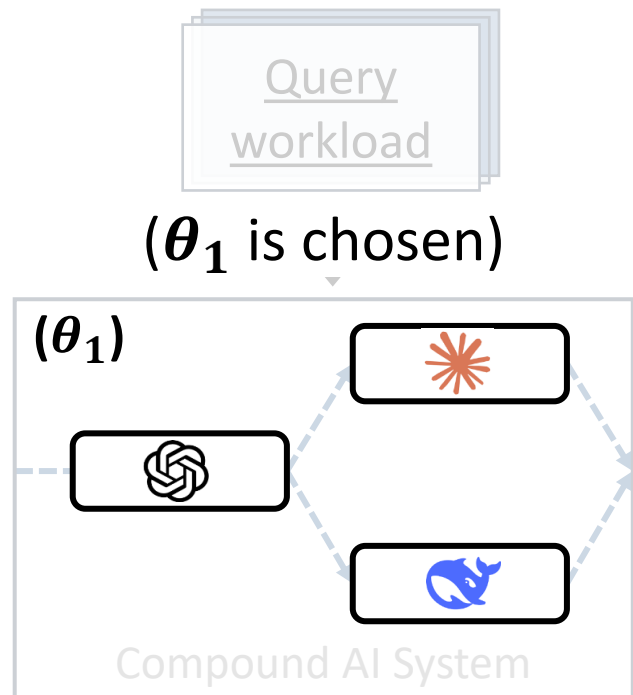
Existing Solutions (Bandit as an example)

- Example: Bandit methods keep estimates for configurations



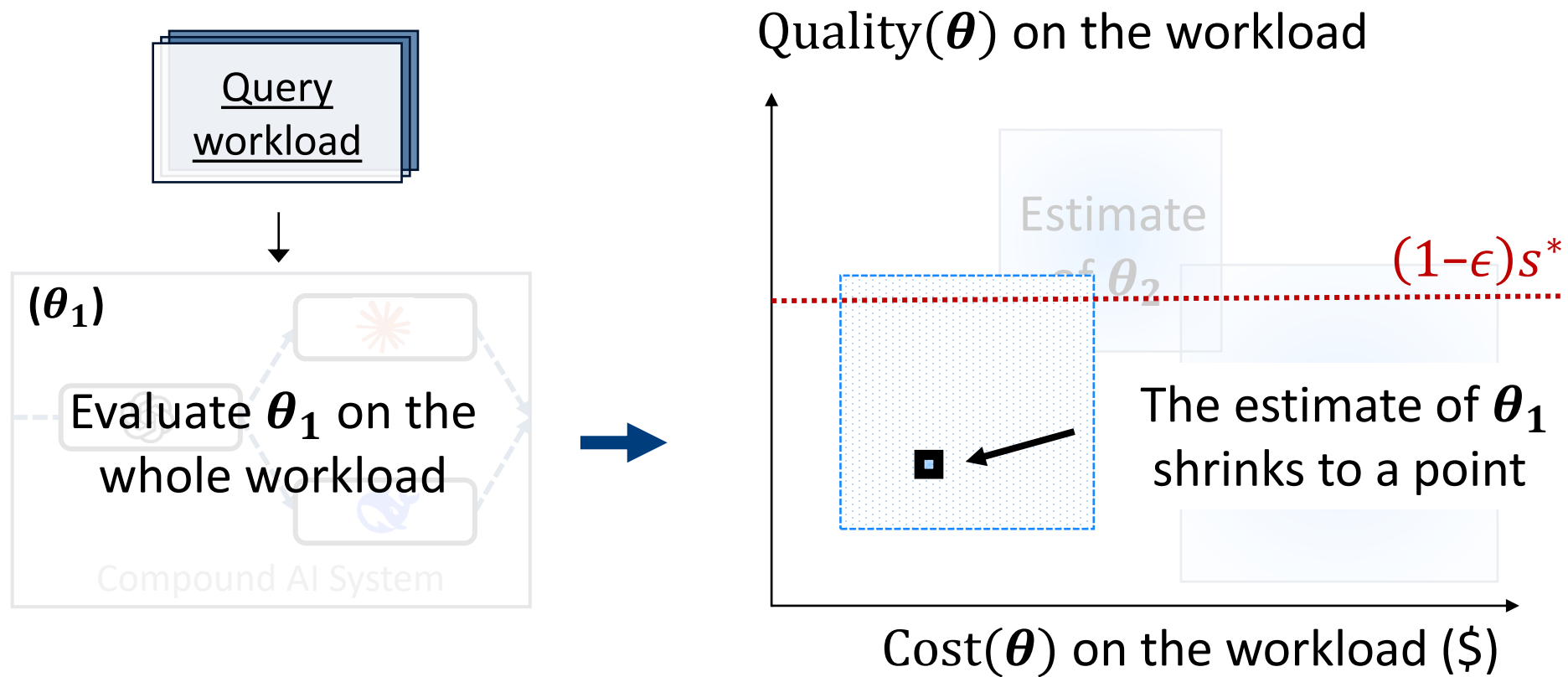
Existing Solutions (Bandit as an example)

- It chooses one with the *best cost estimate* (the leftmost edge)



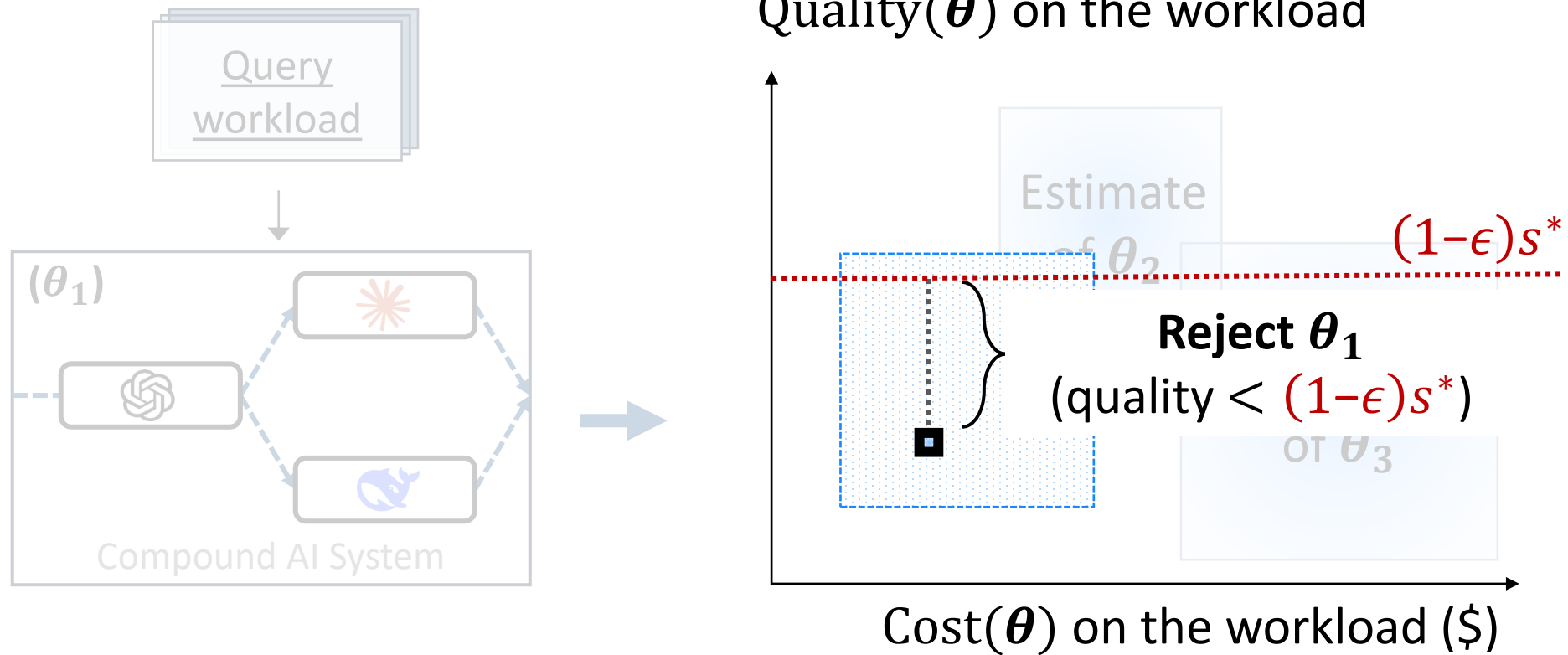
Existing Solutions (Bandit as an example)

- It evaluates θ_1 to get its cost and quality precisely *for the whole workload*



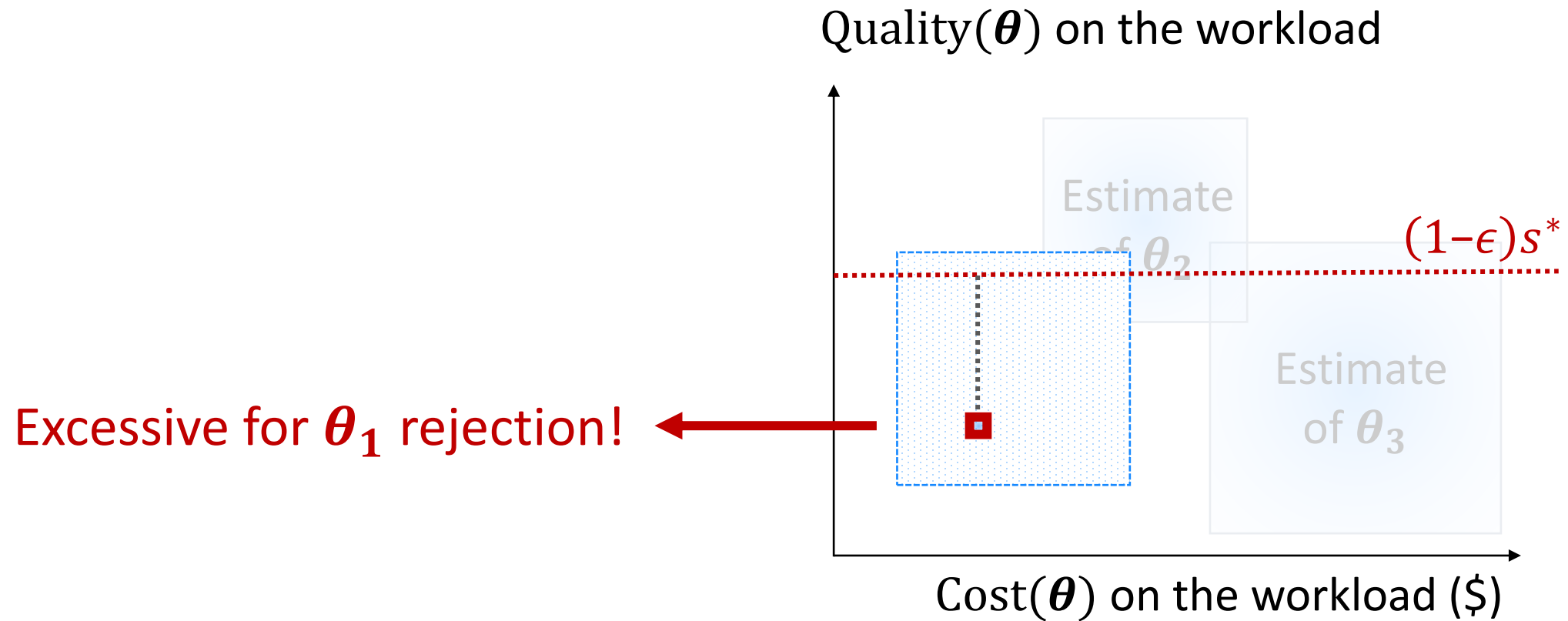
Existing Solutions (Bandit as an example)

- After evaluation, it accepts θ_1 as the next incumbent, or rejects it



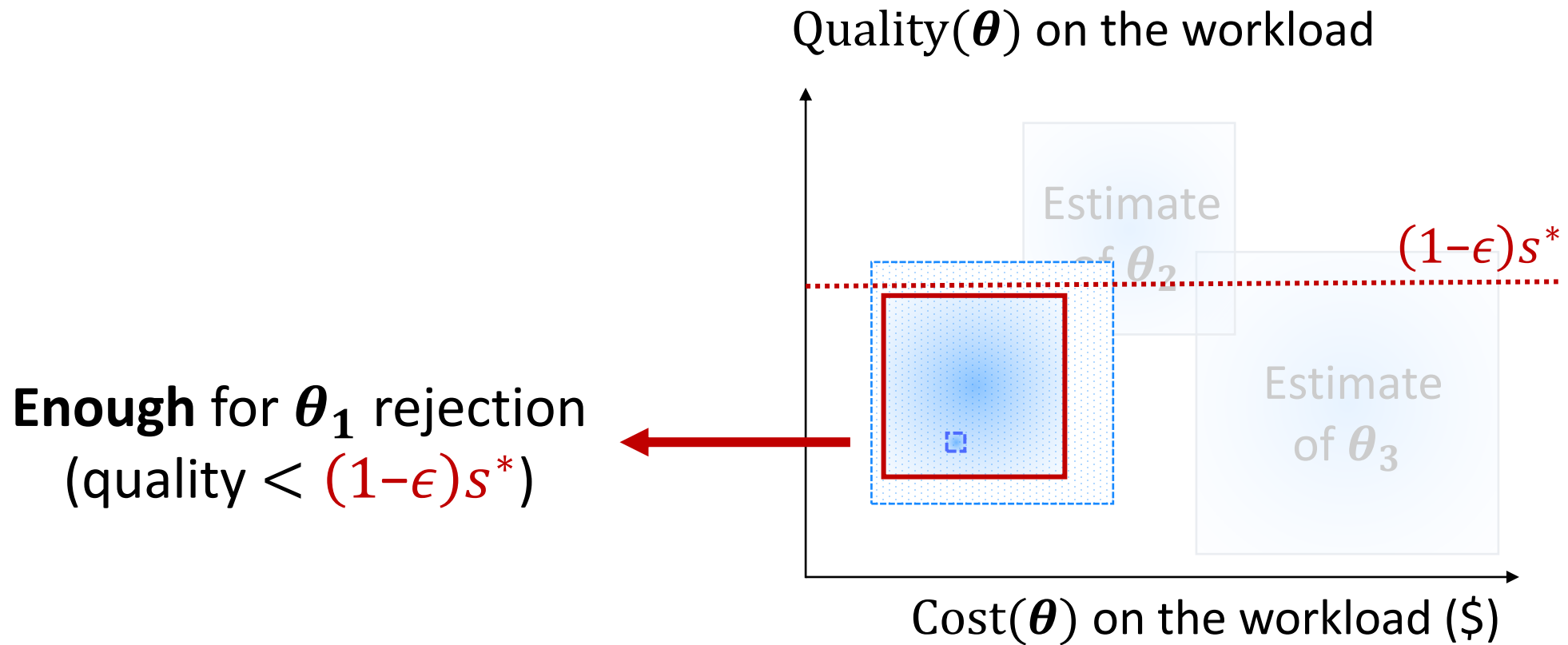
SCOPE (Main idea)

- **Issue:** Existing solutions waste money by unnecessarily shrinking estimates



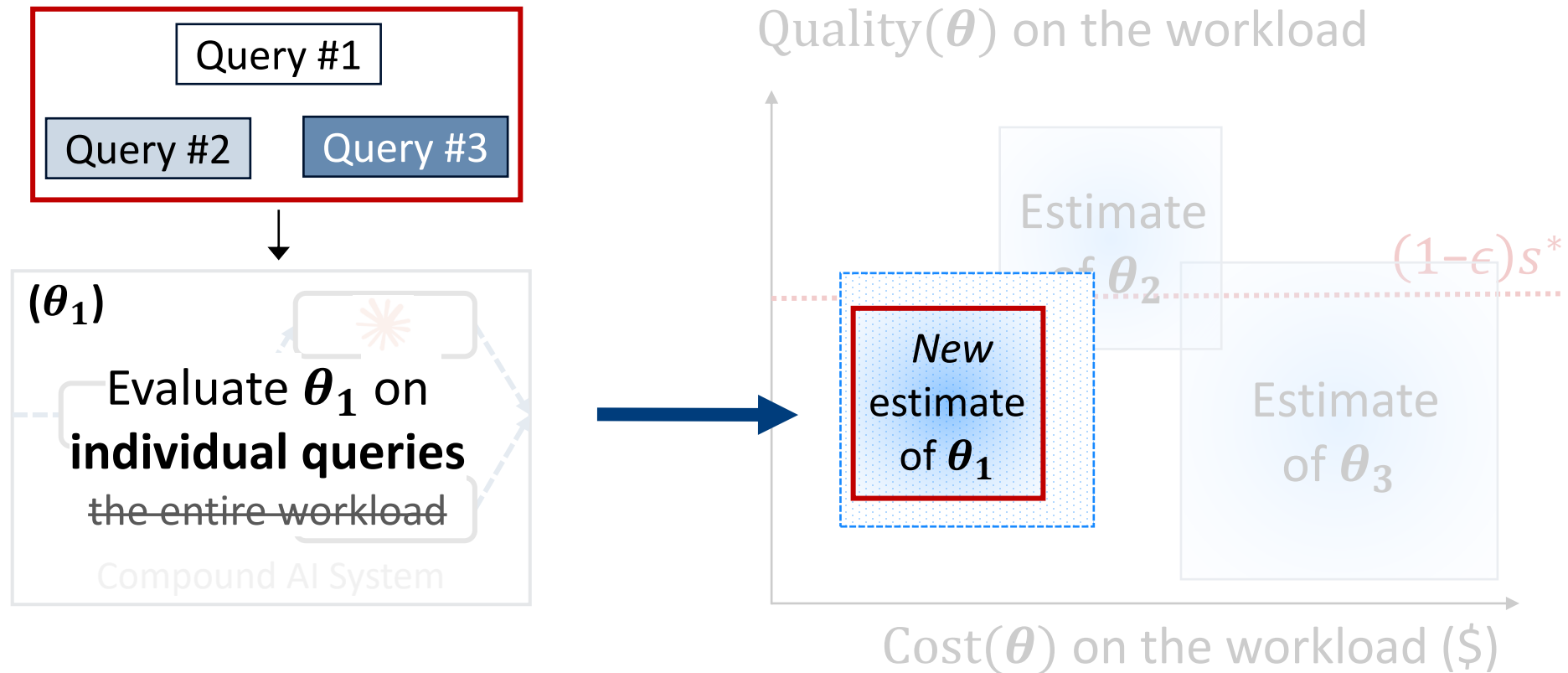
SCOPE (Main idea)

- Instead, a smaller square is just enough here...



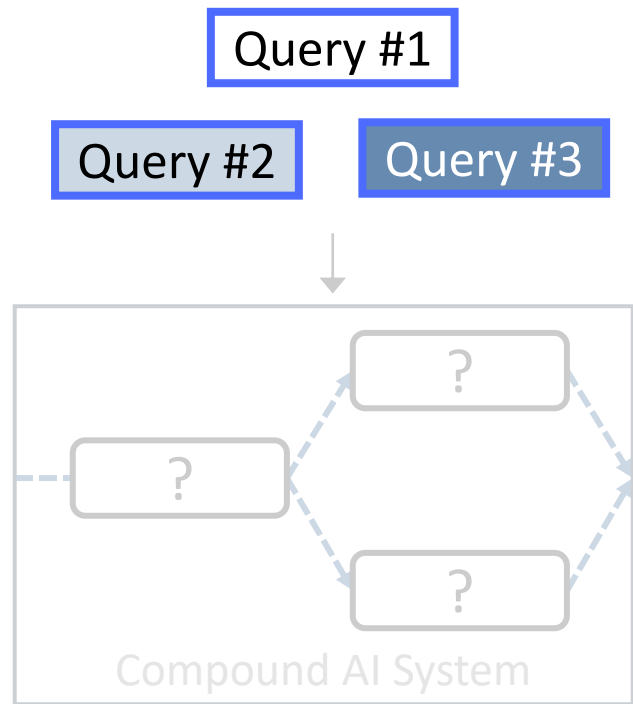
SCOPE (Main idea)

- We get such a smaller square by evaluating the workload *in part* (queries)

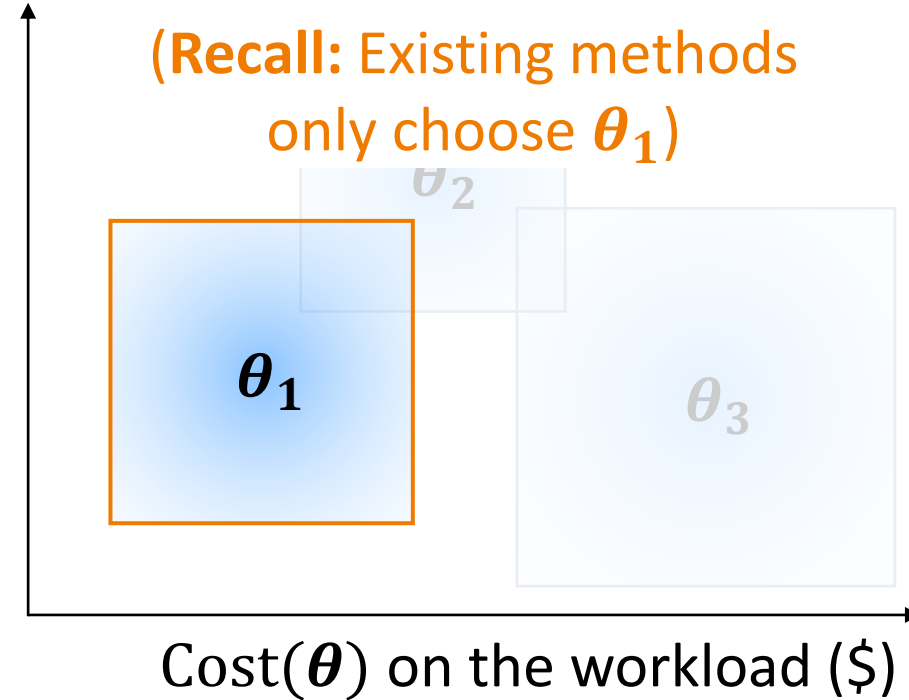


SCOPE (Main idea)

- This is **SCOPE**: it answers *which configuration and query to choose next*

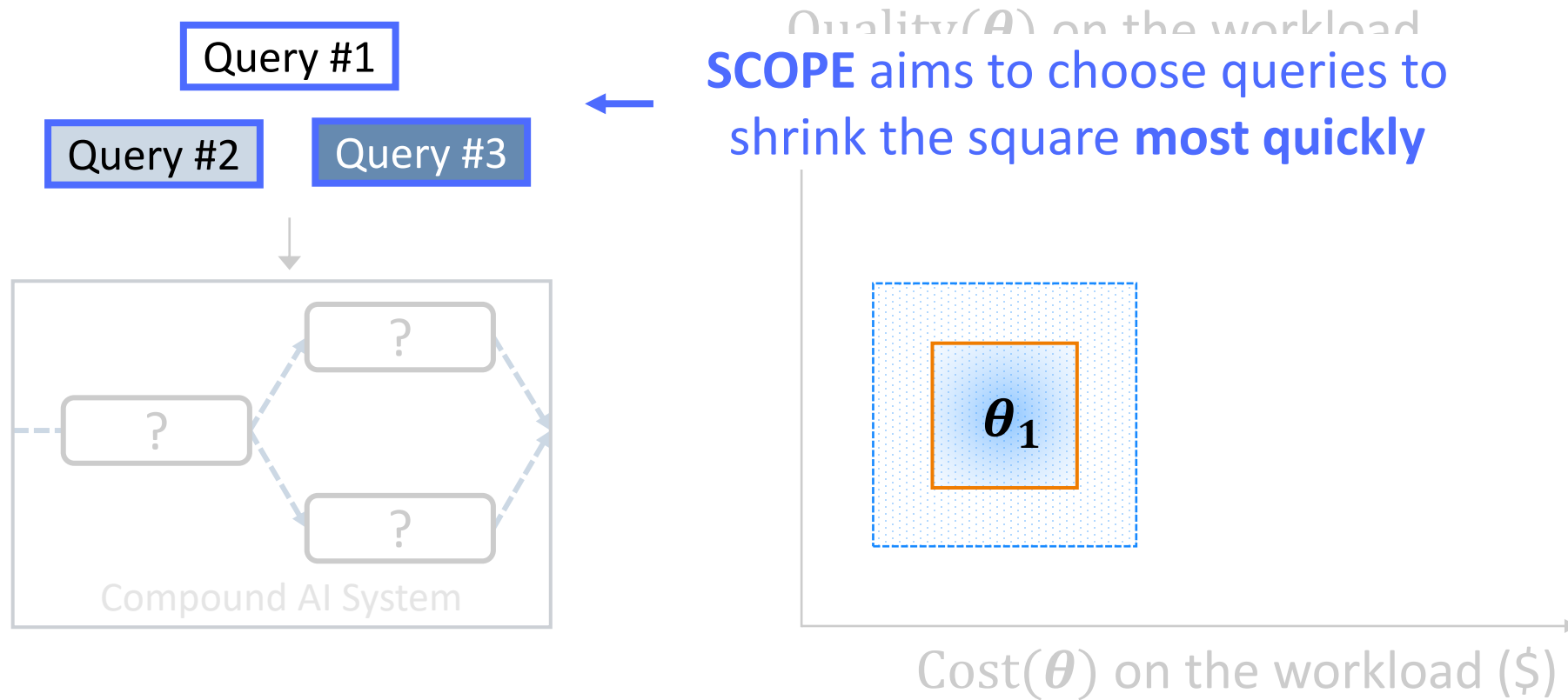


Quality(θ) on the workload



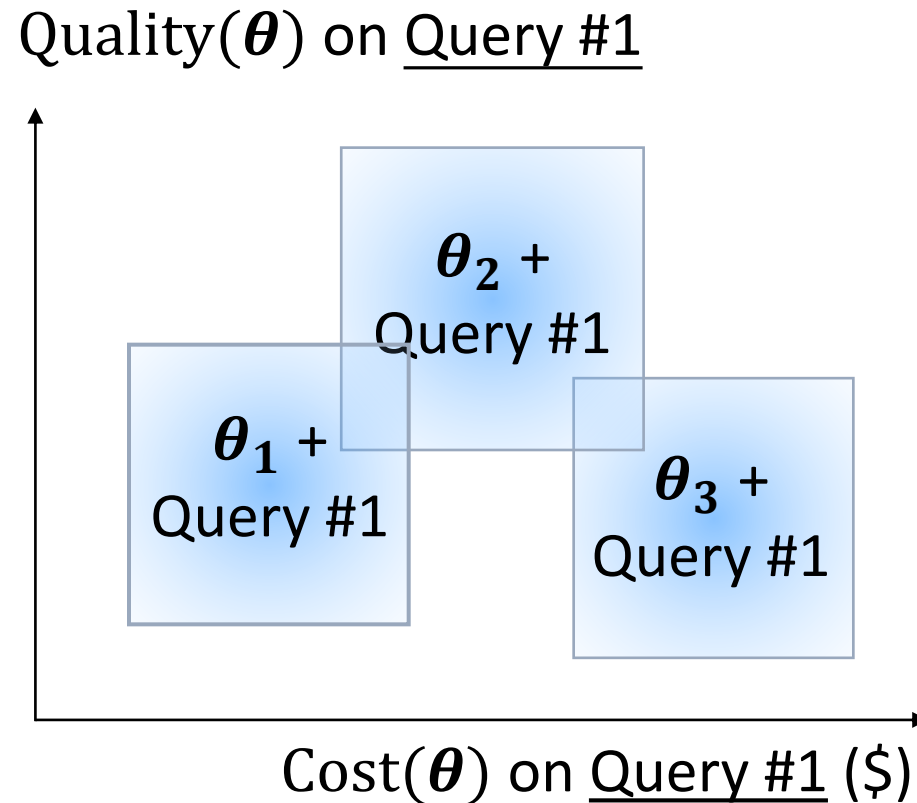
SCOPE (Main idea)

- This is **SCOPE**: it answers *which configuration and query to try next*



SCOPE (in more detail)

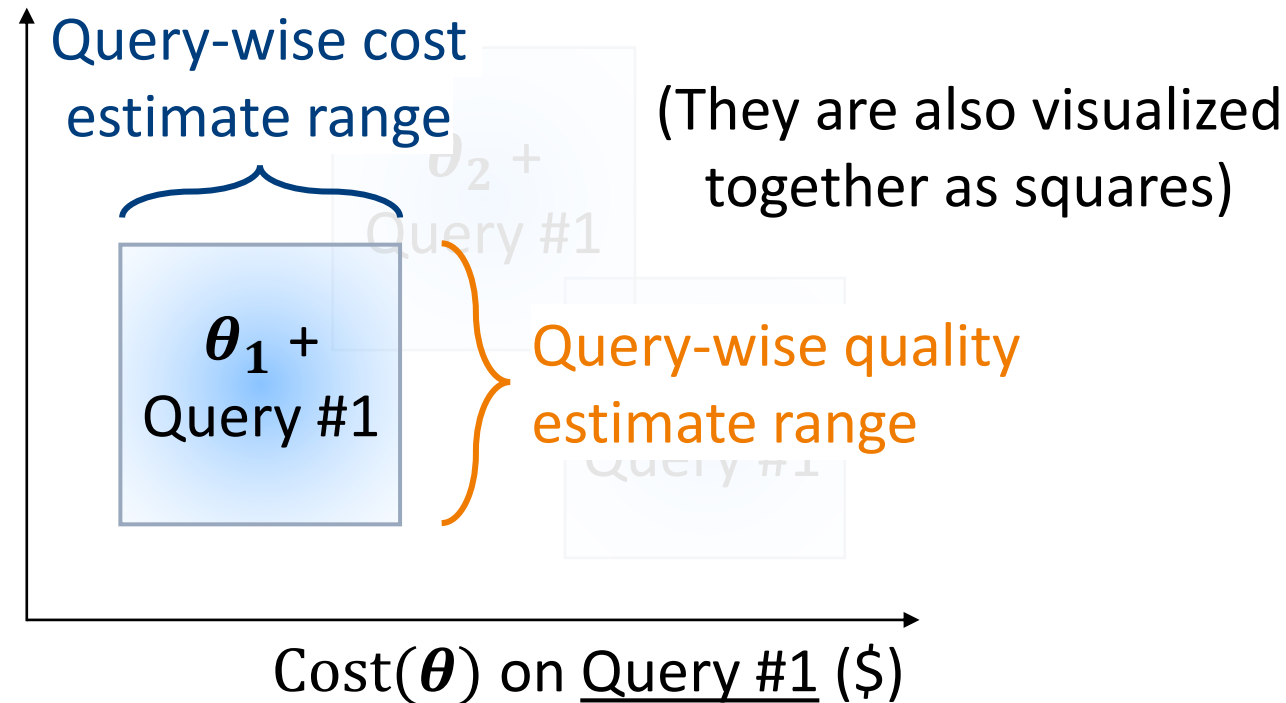
- **Q:** How can we shrink the square most quickly?
- **A:** SCOPE keeps estimates for each query and configuration



SCOPE (in more detail)

- **Q:** How can we shrink the square most quickly?
- **A:** SCOPE keeps estimates for each query and configuration

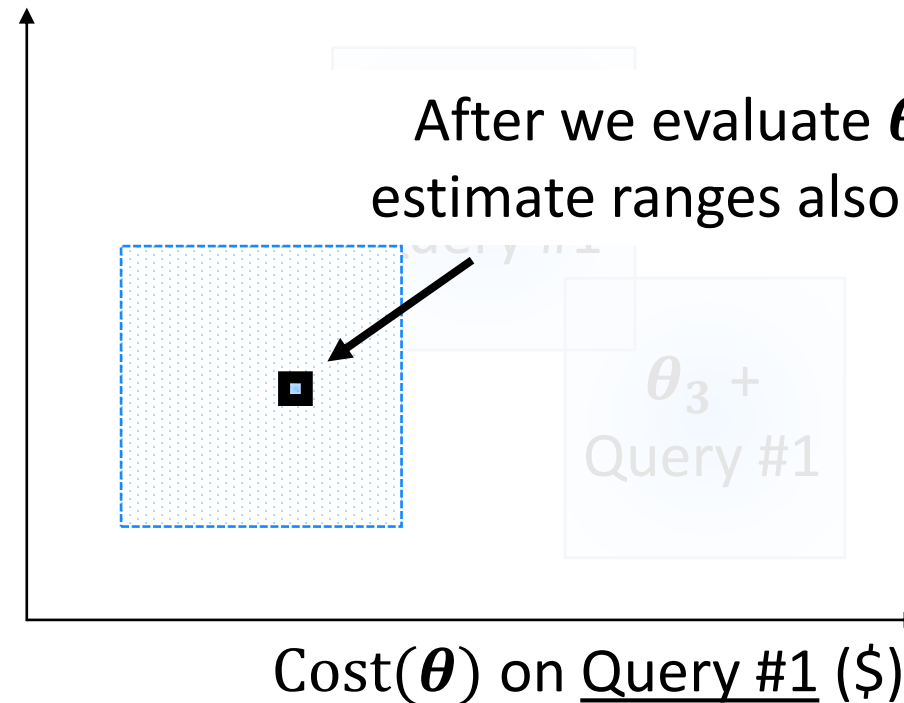
Quality(θ) on Query #1



SCOPE (in more detail)

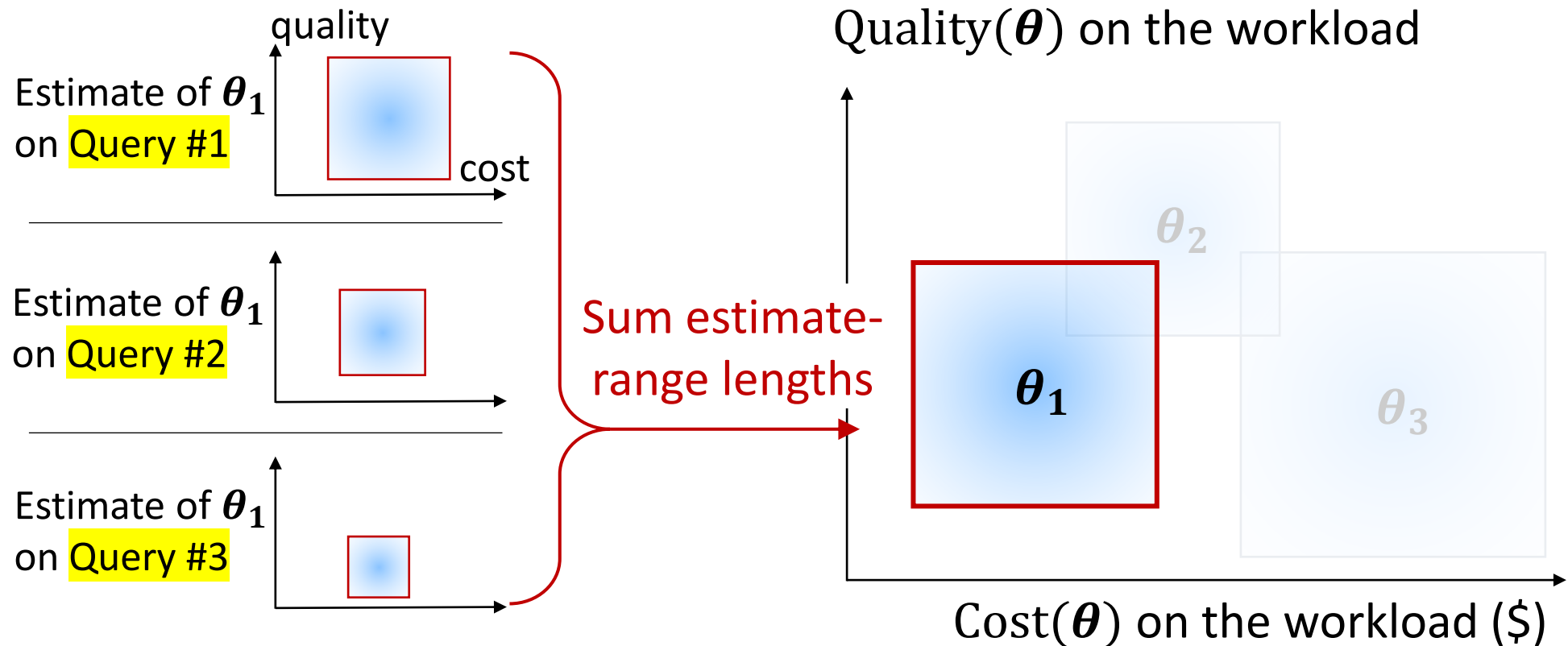
- **Q:** How can we shrink the square most quickly?
- **A:** SCOPE keeps estimates for each query and configuration

Quality(θ) on Query #1



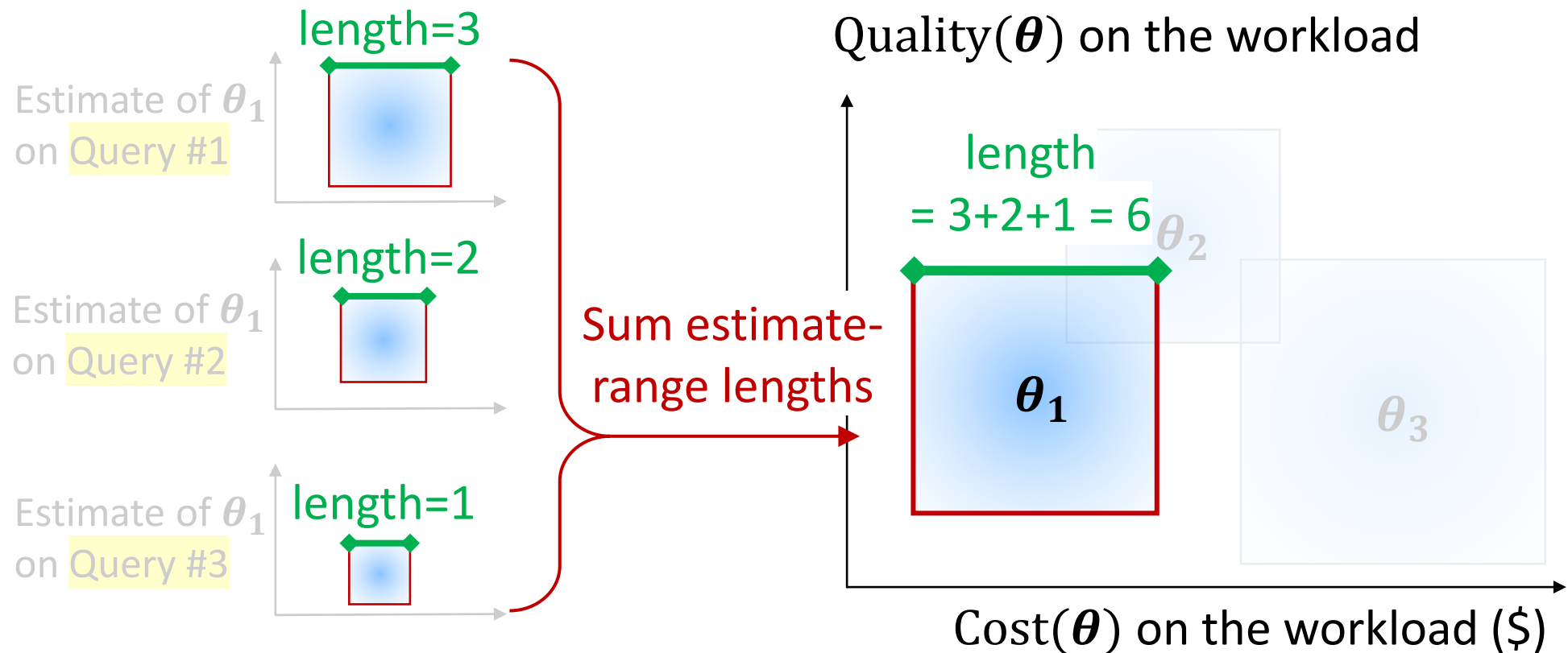
SCOPE (in more detail)

- **Q:** How can we shrink the square most quickly?
- **A:** SCOPE sums query estimate ranges into workload estimate ranges



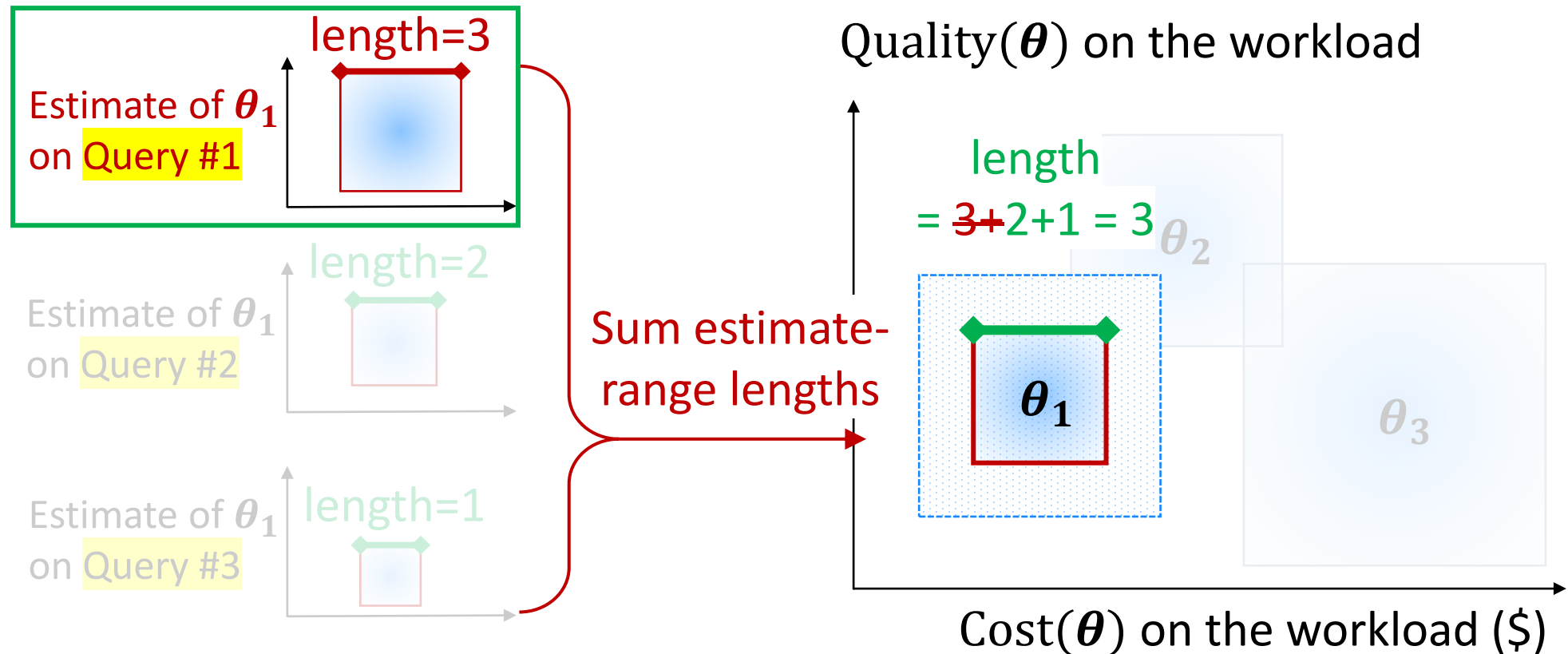
SCOPE (in more detail)

- **Q:** How can we shrink the square most quickly?
- **A:** SCOPE sums query estimate ranges into workload estimate ranges



SCOPE (in more detail)

- **Q:** How can we shrink the square most quickly?
- **A:** Greedily evaluate the query with the **largest side length!**



SCOPE (in more detail)

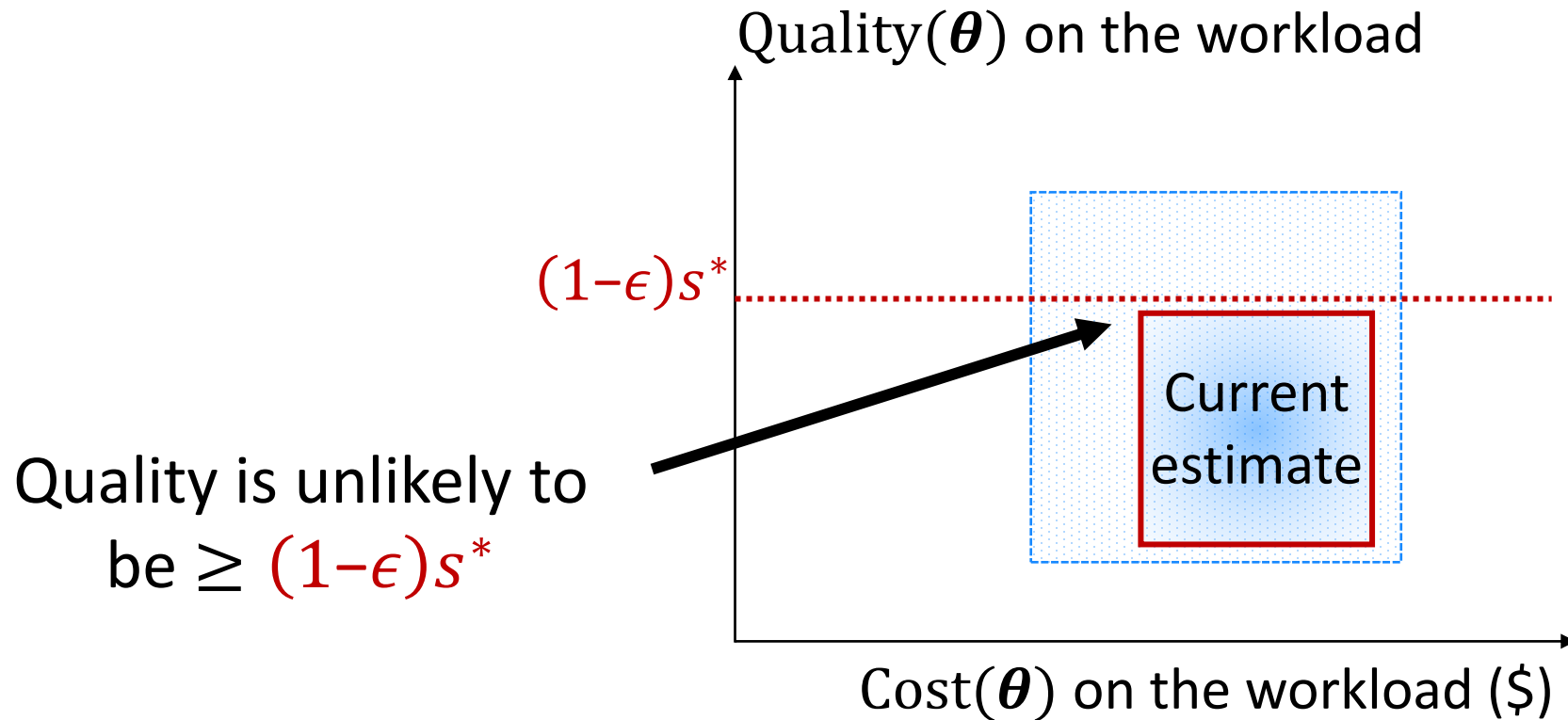
Pseudocode of SCOPE

1. For (trials $t = 1, 2, \dots$ while budget remains):
2. Select a **configuration** $\theta^{(t)}$ (using a way similar to Bandits)
3. While NOT (all queries are evaluated):
4. Select the **query** q with the **largest side length**
5. Evaluate $\theta^{(t)}$ on query q
6. Break the while-loop when either stopping rule is met

SCOPE (in more detail)

SCOPE's two stopping rules

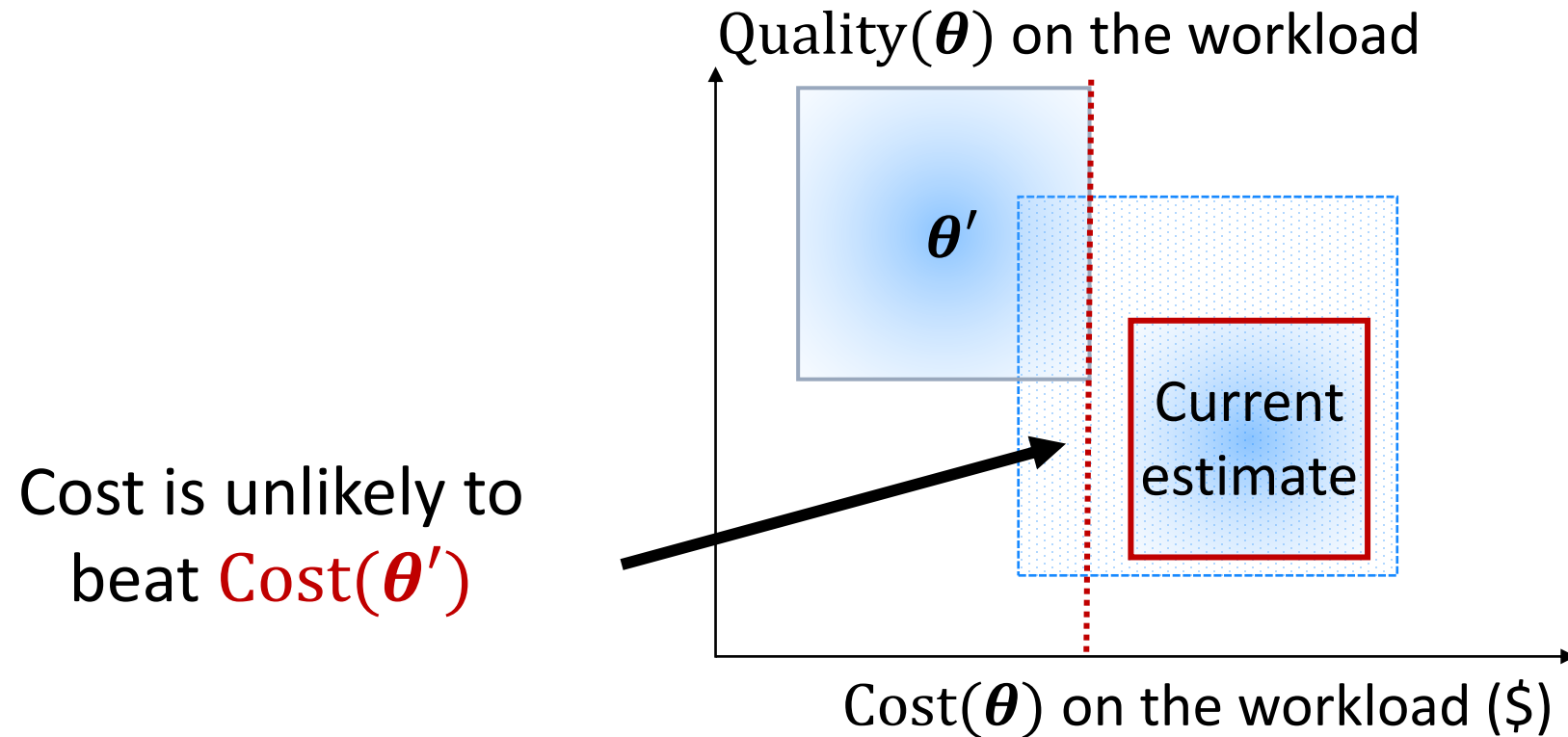
1. The quality estimate falls below $(1-\epsilon)s^*$



SCOPE (in more detail)

SCOPE's two stopping rules

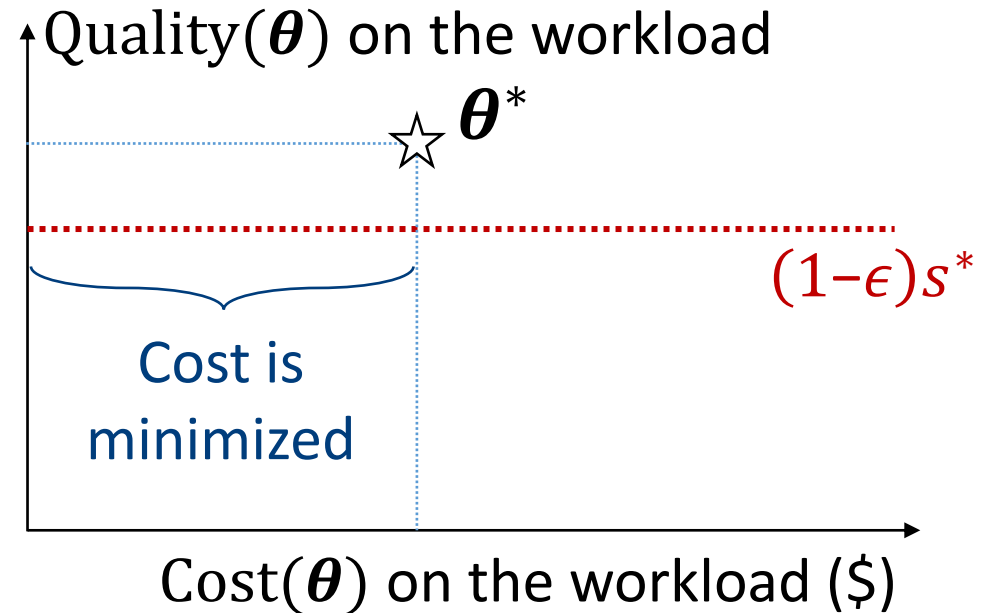
2. The cost estimate is dominated by the incumbent



Theoretical Guarantee

- Let θ^* be the *best feasible* configuration with the lowest cost:

$$\theta^* \in \arg \min_{\theta: \text{Quality}(\theta) \geq (1-\epsilon)s^*} \text{Cost}(\theta)$$



Theoretical Guarantee

- Let θ^* be the *best feasible* configuration with the lowest cost:

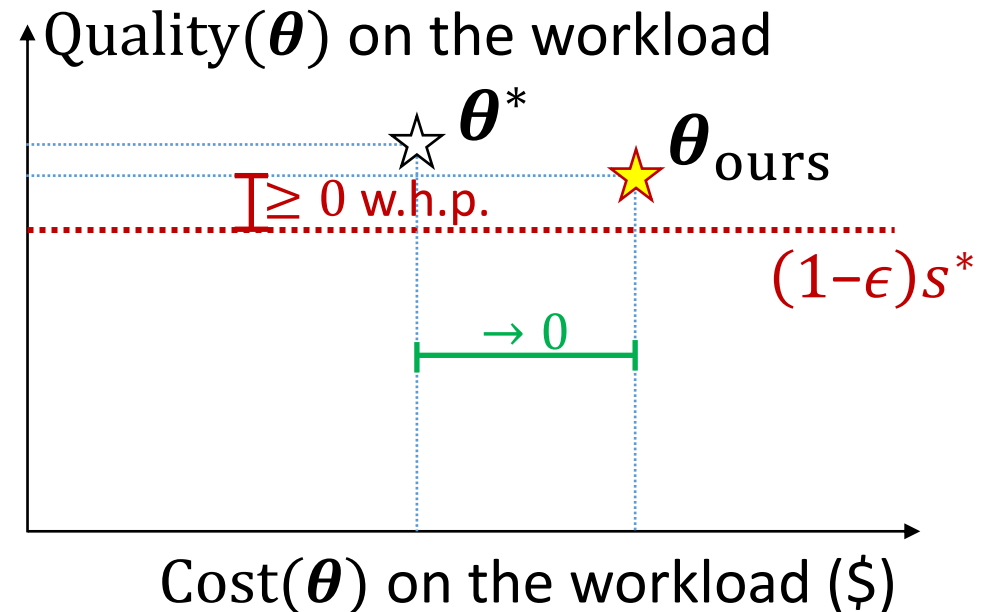
$$\theta^* \in \arg \min_{\theta: \text{Quality}(\theta) \geq (1-\epsilon)s^*} \text{Cost}(\theta)$$

- SCOPE's output (θ_{ours}) satisfies:

1. $\text{Quality}(\theta_{\text{ours}}) \geq (1-\epsilon)s^*$ w.h.p.

2. $|\text{Cost}(\theta_{\text{ours}}) - \text{Cost}(\theta^*)| \rightarrow 0$

when we increase # of trials



Experiments

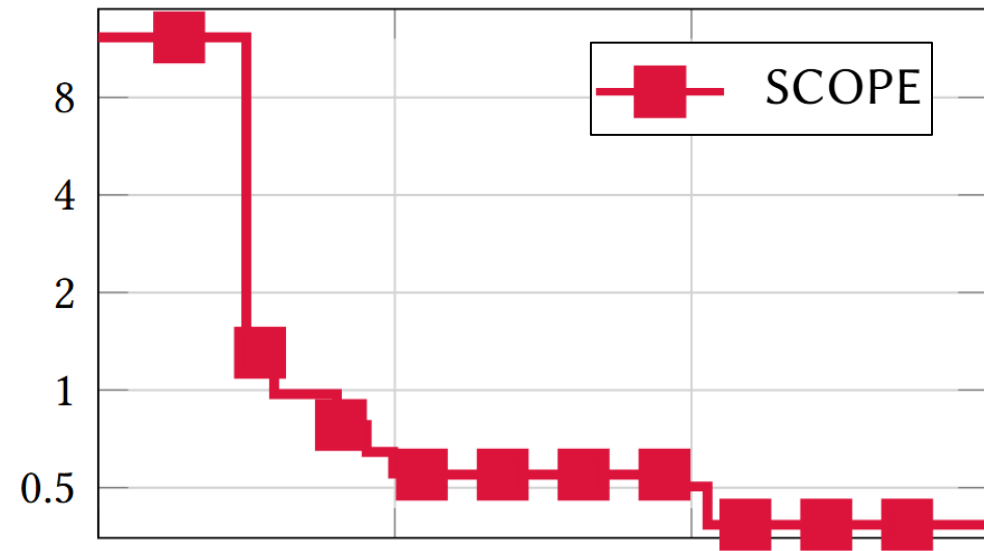
- 3 systems: Text-to-SQL, Data transformation, Data imputation
- **23 candidate LLMs** ($\Rightarrow$ up to $\sim 6,400,000$ possible configurations)
- One workload for each system; $\epsilon = 0.01$

Experiments

- We compare how fast a curve drops
- We will see something like this:

X-axis: budget (\$) for each method

Y-axis: cost (\$) of best found configuration
(subject to quality $\geq (1-\epsilon)s^*$; *log-scale*)

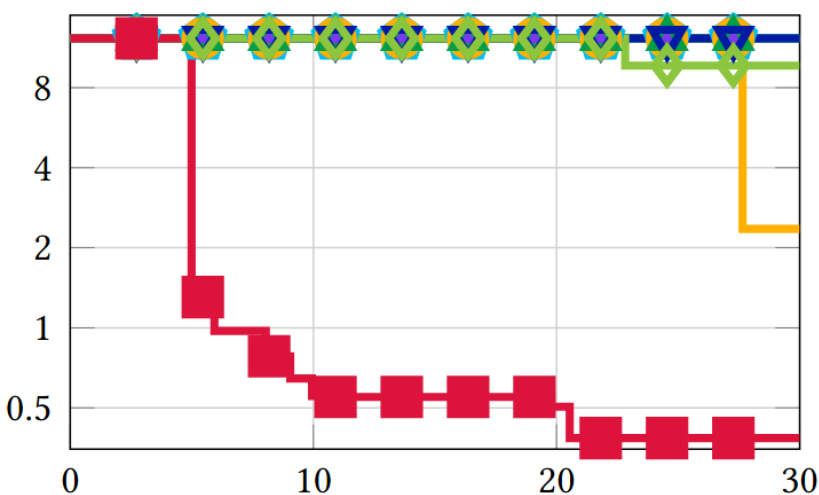


Experiments

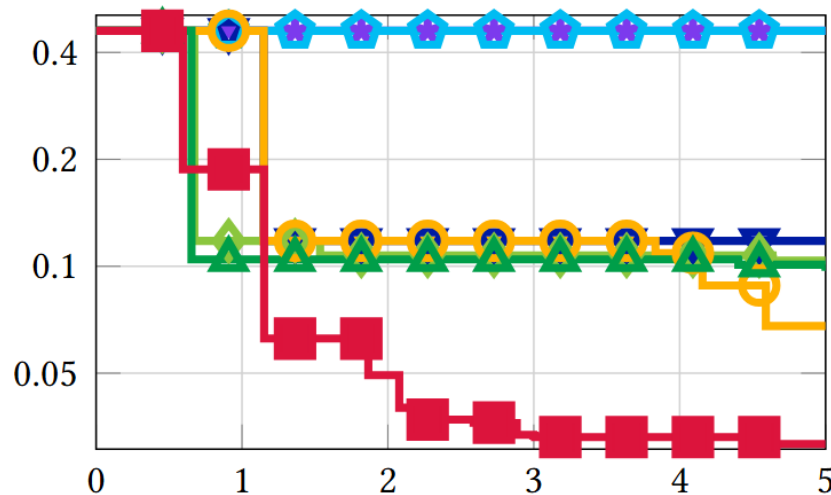
- We compare how fast a curve drops

—■— SCOPE —○— cEI —◇— CONFIG —△— LLAMBO —◇— Abacus —*— LLMSelector —▽— SafeOpt

Y-axis: cost (\$) of best found configuration (subject to quality $\geq s_0$; *log-scale*)

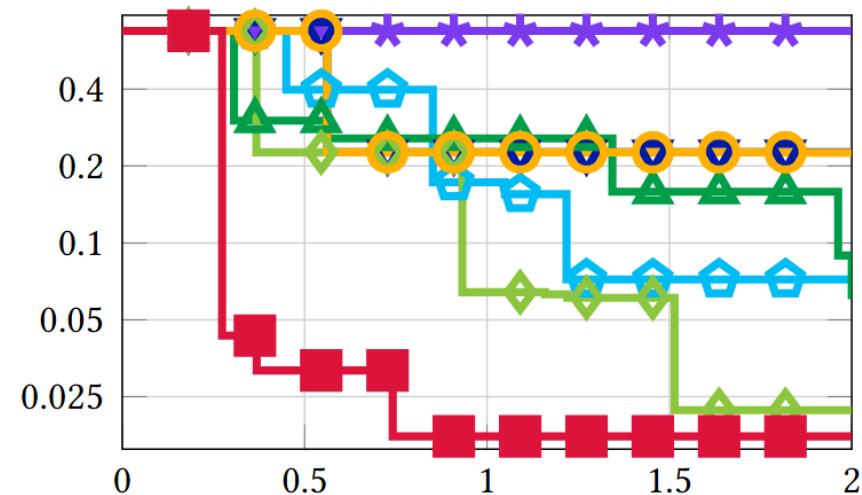


Text-to-SQL



X-axis: budget (\$) for each method

Data transformation



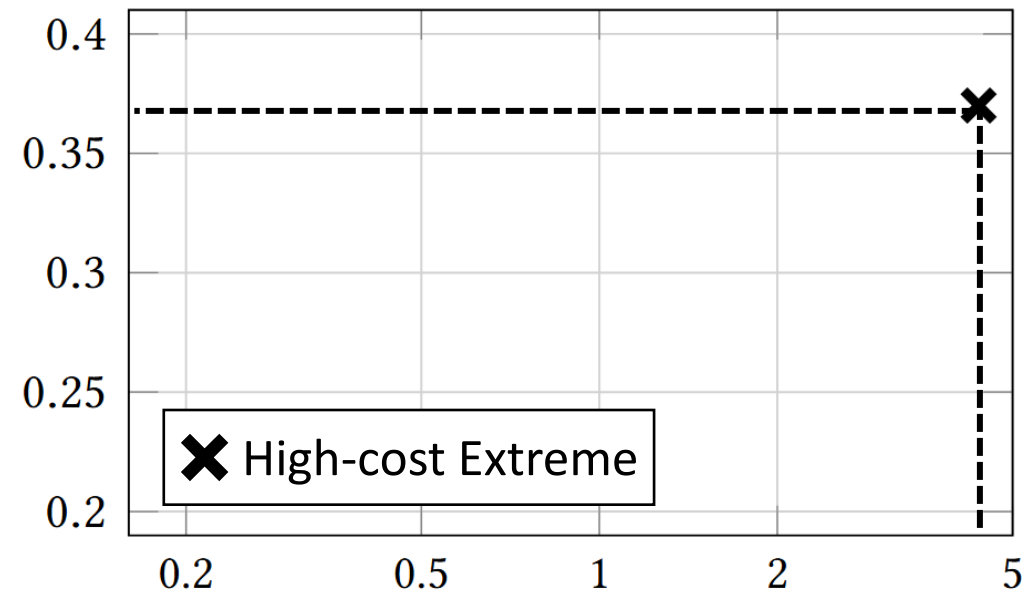
Data imputation

Experiments (Generalization)

- We re-run the system with final configurations on **another workload**
- We will see something like this

X-axis: cost (\$) on **another workload**
(*log-scale*)

Y-axis: quality (acc.) on **another workload**

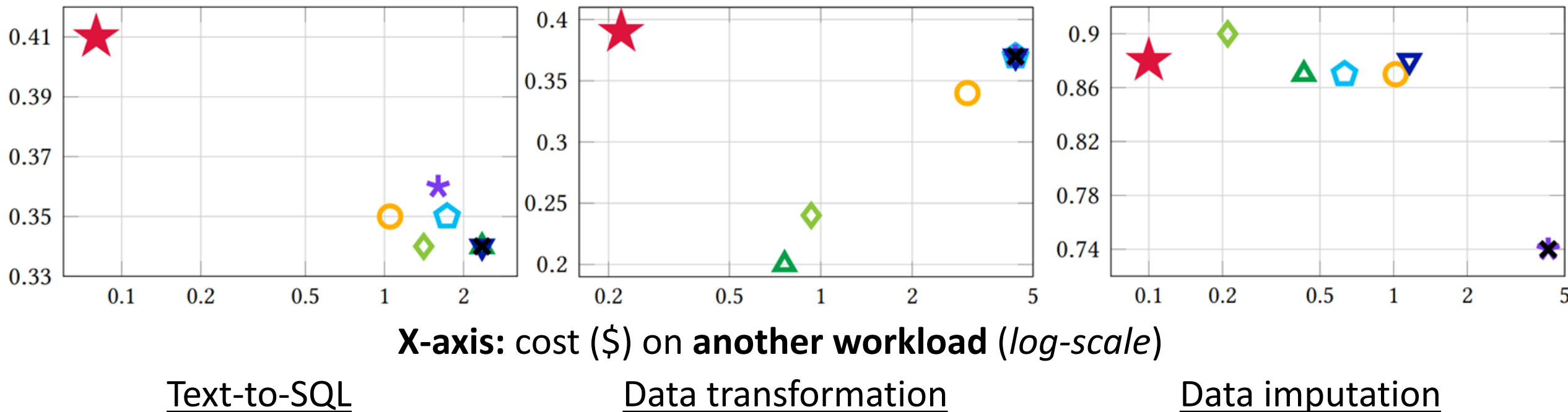


Experiments (Generalization)

- We mainly compare with **✕ High-cost Extreme**

★ SCOPE ○ cEI ◇ CONFIG ▲ LLAMBO ⬡ Abacus ✱ LLMSelector ▼ SafeOpt

Y-axis: quality (acc.) on another workload

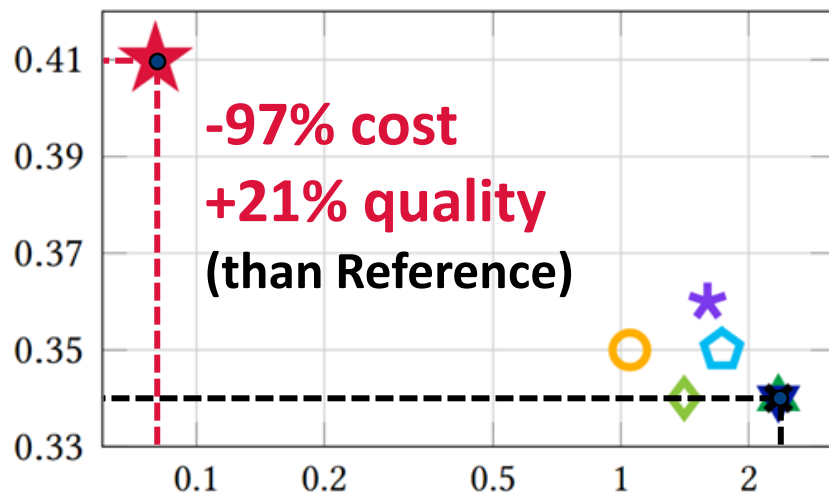


Experiments (Generalization)

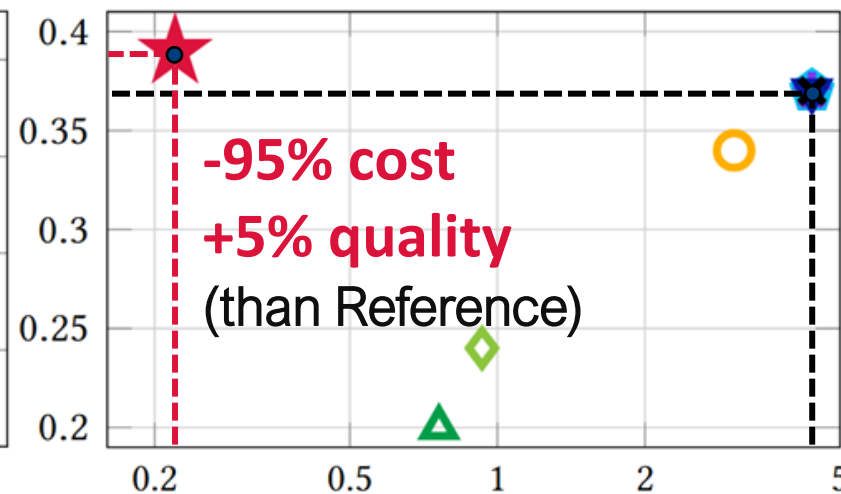
- We mainly compare with **✕ High-cost Extreme** (a clearer view)

★ SCOPE ○ cEI ◇ CONFIG ▲ LLAMBO ◑ Abacus ✱ LLMSelector ▼ SafeOpt

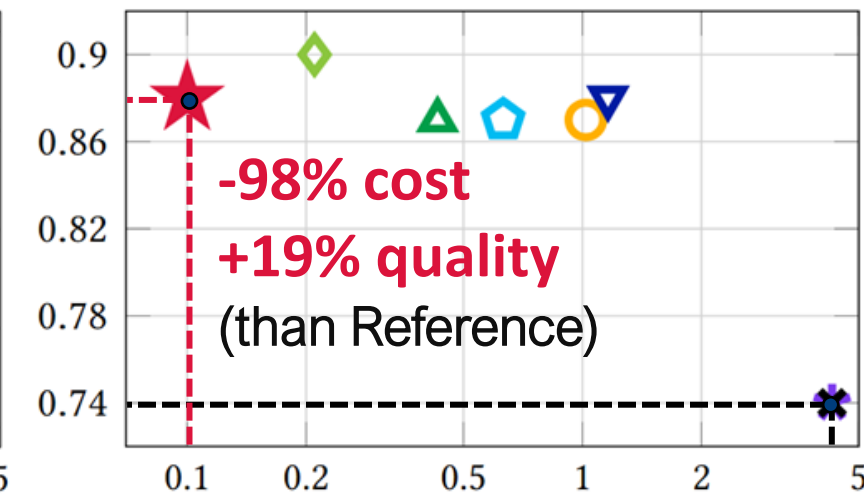
Y-axis: quality (acc.) on another workload



Text-to-SQL



Data transformation



Data imputation

More in the paper

- How does **SCOPE** select a configuration ($\theta^{(t)}$)?
- Details of estimation techniques
 - What confidence bounds are used?
 - How do configuration (and query) estimates relate to each other?
- How does **SCOPE** cope with LLM randomness?
- More experimental results

Future Work

- Multi-objective optimization
(can we optimize both cost and quality at the same time?)
- Optimization on dynamic and agentic systems
- Etc.

Thanks!

Code: github.com/waetr/SCOPELLM-optimizer/

Email: yiqian@comp.nus.edu.sg