Efficient Capability-Aware LLM Systems: Capability Modeling, Routing, and Load Balancing

Fangzhou Wu

University of Wisconsin–Madison

Increasing Cost in LLM Serving

- Large Language Models (LLMs) are increasingly embedded in diverse domains.
- Growing query volume imposes substantial inference costs.

Increasing Cost in LLM Serving

- Large Language Models (LLMs) are increasingly embedded in diverse domains.
- Growing query volume imposes substantial inference costs.
- **Efficiently delivering high-quality service under tight token budgets is critical for LLM providers.**

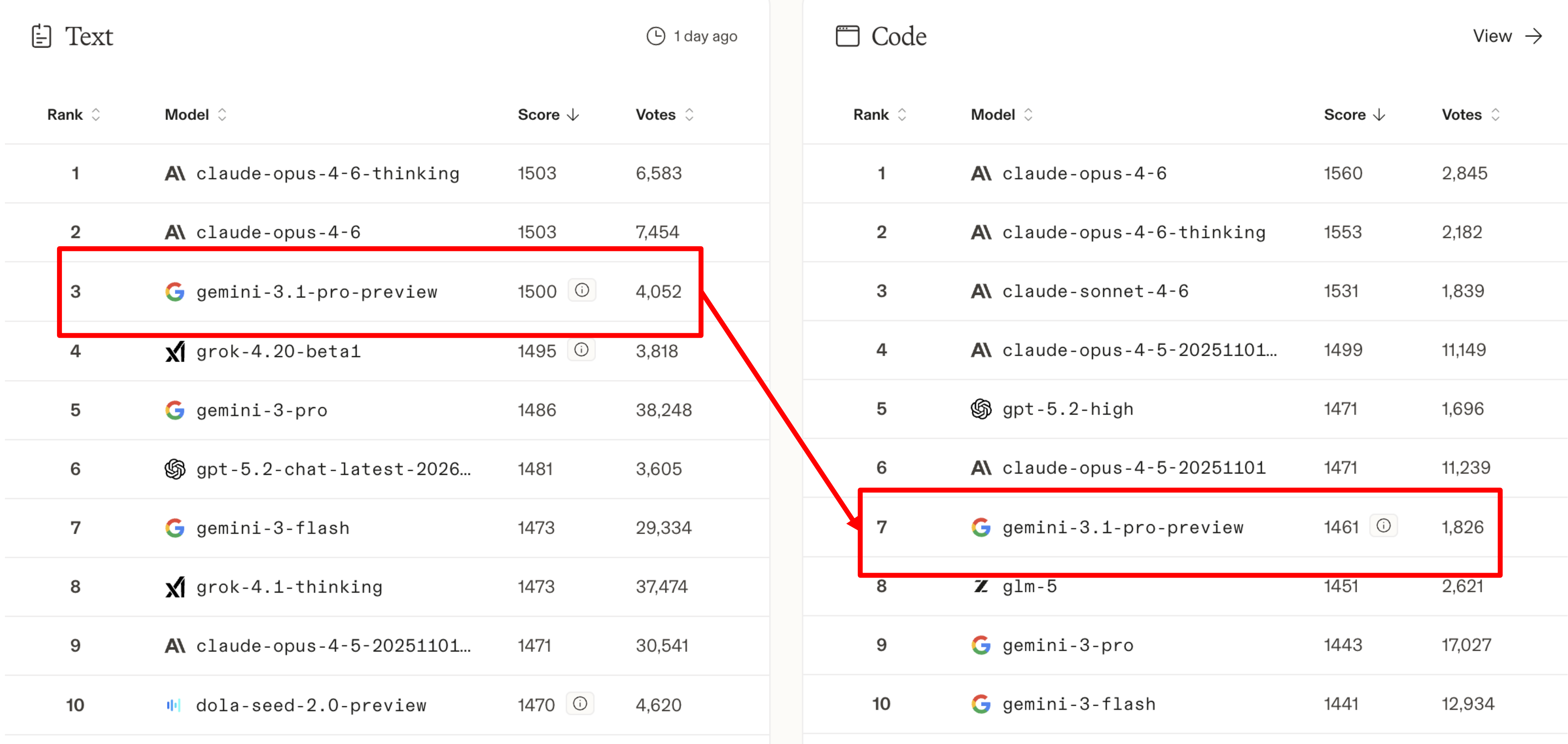
Large Language Model with Diverse Capabilities

- LLMs trained on extensive corpora exhibit diverse capabilities (e.g., math, coding).

Benchmark		Gemini 3.1 Pro Thinking (High)	Gemini 3 Pro Thinking (High)	Sonnet 4.6 Thinking (Max)	Opus 4.6 Thinking (Max)	GPT-5.2 Thinking (xhigh)	GPT-5.3-Codex Thinking (xhigh)
Humanity's Last Exam Academic reasoning (full set, text + MM)	No tools	**44.4%**	37.5%	33.2%	40.0%	34.5%	—
	Search (blocklist) + Code	51.4%	45.8%	49.0%	**53.1%**	45.5%	—
ARC-AGI-2 Abstract reasoning puzzles	ARC Prize Verified	**77.1%**	31.1%	58.3%	68.8%	52.9%	—
GPQA Diamond Scientific knowledge	No tools	**94.3%**	91.9%	89.9%	91.3%	92.4%	—
Terminal-Bench 2.0 Agentic terminal coding	Terminus-2 harness	**68.5%**	56.9%	59.1%	65.4%	54.0%	64.7%
	Other best self-reported harness	—	—	—	—	62.2% (Codex)	**77.3%** (Codex)
SWE-Bench Verified Agentic coding	Single attempt	80.6%	76.2%	79.6%	**80.8%**	80.0%	—
SWE-Bench Pro (Public) Diverse agentic coding tasks	Single attempt	54.2%	43.3%	—	—	55.6%	**56.8%**

Large Language Model with Diverse Capabilities

- LLMs trained on extensive corpora exhibit diverse capabilities (e.g., math, coding).
- **Different LLMs excel in different tasks.**



Text — 1 day ago

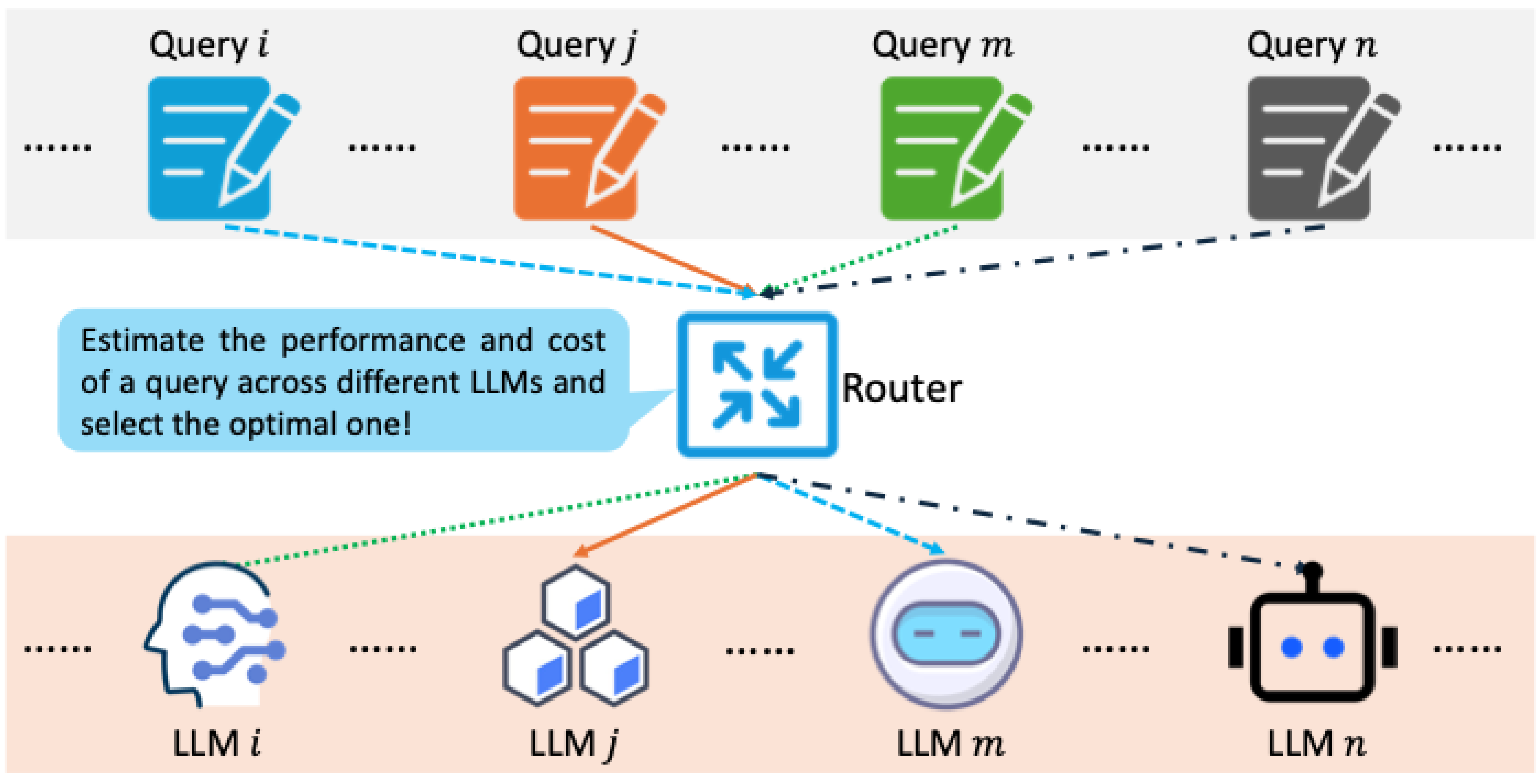
Rank	Model	Score	Votes
1	claude-opus-4-6-thinking	1503	6,583
2	claude-opus-4-6	1503	7,454
3	gemini-3.1-pro-preview	1500	4,052
4	grok-4.20-beta1	1495	3,818
5	gemini-3-pro	1486	38,248
6	gpt-5.2-chat-latest-2026...	1481	3,605
7	gemini-3-flash	1473	29,334
8	grok-4.1-thinking	1473	37,474
9	claude-opus-4-5-20251101...	1471	30,541
10	dola-seed-2.0-preview	1470	4,620

Code — View →

Rank	Model	Score	Votes
1	claude-opus-4-6	1560	2,845
2	claude-opus-4-6-thinking	1553	2,182
3	claude-sonnet-4-6	1531	1,839
4	claude-opus-4-5-20251101...	1499	11,149
5	gpt-5.2-high	1471	1,696
6	claude-opus-4-5-20251101	1471	11,239
7	gemini-3.1-pro-preview	1461	1,826
8	glm-5	1451	2,621
9	gemini-3-pro	1443	17,027
10	gemini-3-flash	1441	12,934

LLM Routing

- LLM routing offers a cost-efficient solution by directing each query to the optimal LLM that best balances response quality and inference cost.



Formal Problem Formulation

- Consider an LLM-serving system deployed with M LLMs with each LLM i is assigned a token budget B_i.
- Sending query j to LLM i yields quality score d_{ij} and cost g_{ij}.

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Formal Problem Formulation

- Consider an LLM-serving system deployed with M LLMs with each LLM i is assigned a token budget B_i.
- Sending query j to LLM i yields quality score d_{ij} and cost g_{ij}.
- **Objective**: Route a set of Q queries with a routing strategy $x(\cdot)$ that maximizes the overall quality of responses.

$$
\begin{aligned}
\max \sum_{j \in Q} \sum_{i \in [M]} & d_{ij} x_{ij} \\
\text{s.t.} \sum_{j} & g_{ij} x_{ij} \leq B_i \ \text{ for all } i, \\
\sum_{i} & x_{ij} \leq 1 \ \text{ for all } j, \\
& x_{ij} \in \{0, 1\}
\end{aligned}
\tag{1}
$$

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Formal Problem Formulation

- Consider an LLM-serving system deployed with M LLMs with each LLM i is assigned a token budget B_i.
- Sending query j to LLM i yields quality score d_{ij} and cost g_{ij}.
- **Objective**: Route a set of Q queries with a routing strategy $x(\cdot)$ that maximizes the overall quality of responses.

$$
\begin{aligned}
\max \sum_{j \in Q} \sum_{i \in [M]} & \, d_{ij} x_{ij} \\
\text{s.t.} \sum_{j} g_{ij} x_{ij} & \leq B_i \quad \text{for all } i, \\
\sum_{i} x_{ij} & \leq 1 \quad \text{for all } j, \\
x_{ij} & \in \{0, 1\}
\end{aligned}
\tag{1}
$$

Solving this MILP online is non-trivial: key challenges arise in practice!

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Challenges

- **Inaccessible ground truth performance and cost**.
    - For any query j, the true quality score d_{ij} and cost g_{ij} are **unavailable** without accessing the actual LLMs.

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Challenges

- **Inaccessible ground truth performance and cost**.
  - For any query j, the true quality score d_{ij} and cost g_{ij} are **unavailable** without accessing the actual LLMs.

- **Deployment scalability**.
  - LLM deployment configurations may vary across different environments. Routing algorithm must be adaptive to these variations while minimizing adaptation overhead.

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

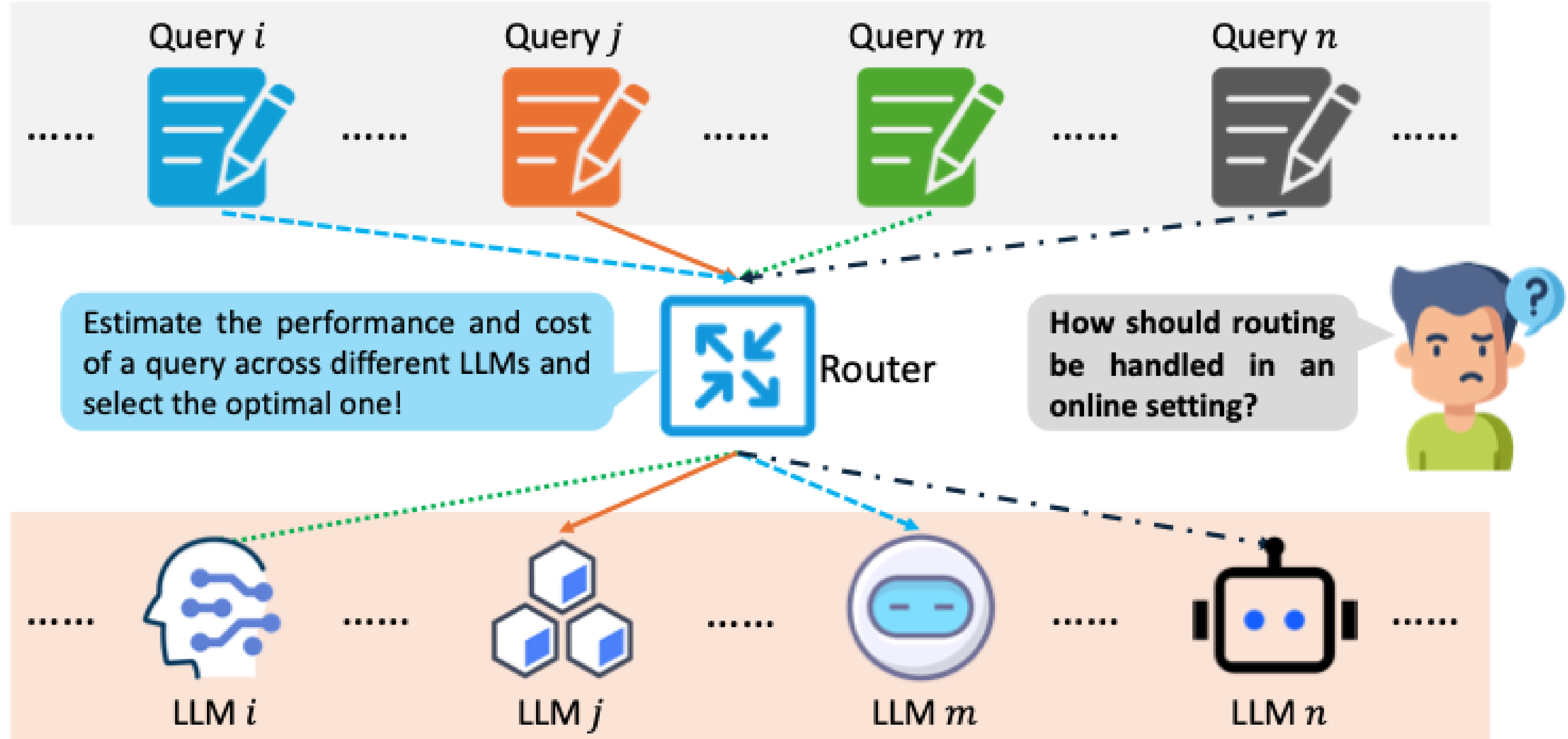
Challenges

- **Inaccessible ground truth performance and cost**.
  - For any query j, the true quality score d_{ij} and cost g_{ij} are **unavailable** without accessing the actual LLMs.
- **Deployment scalability**.
  - LLM deployment configurations may vary across different environments. Routing algorithm must be adaptive to these variations while minimizing adaptation overhead.
- **Sequential query arrival under uncertainty**.
  - In practice, queries arrive sequentially rather than simultaneously and must be routed without knowledge of future queries.

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Motivation

- Can we design a **training-free online routing algorithm** that still achieves a near-optimal cumulative performance?



Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

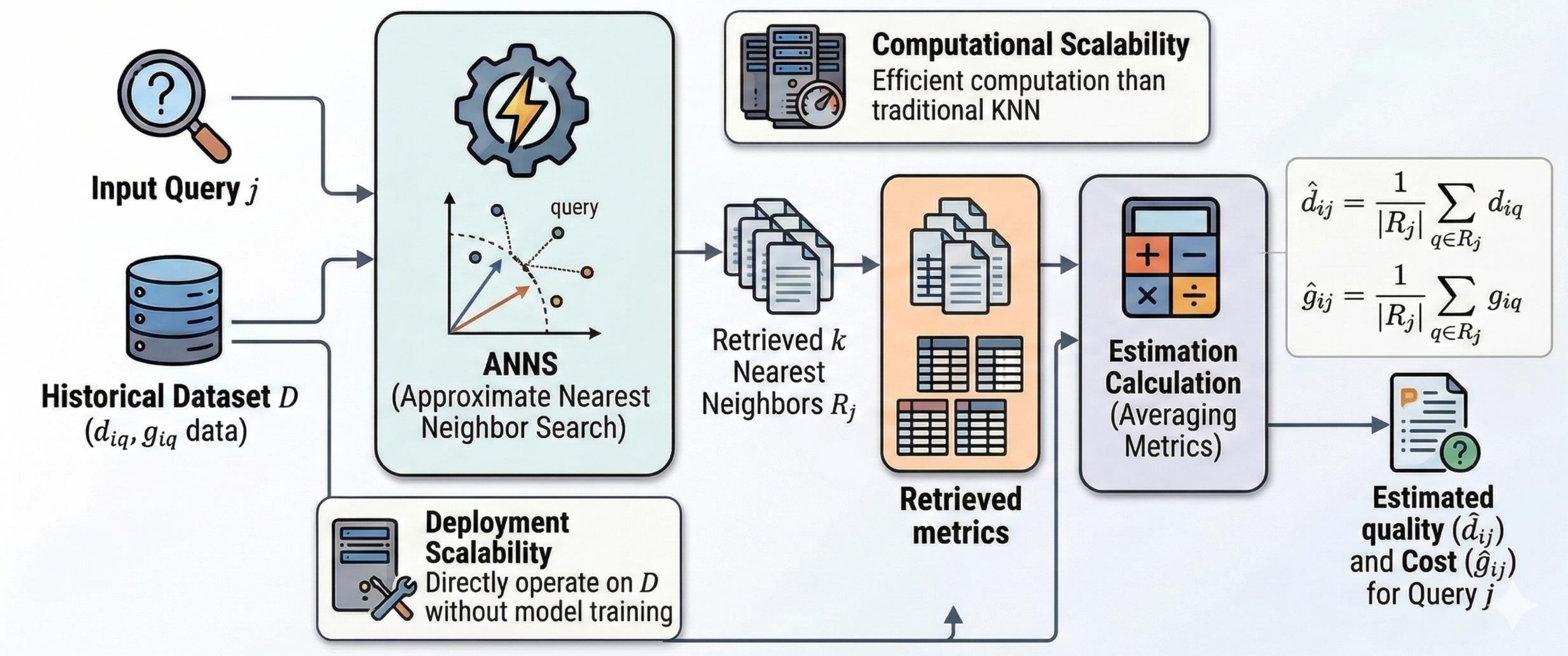
Efficient Quality and Cost Estimation

- For each query, we employ Approximate Nearest Neighbor Search (ANNS) to efficiently estimate the response quality and cost for each deployed LLM using a historical dataset D.

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Efficient Quality and Cost Estimation

- For each query, we employ Approximate Nearest Neighbor Search (ANNS) to efficiently estimate the response quality and cost for each deployed LLM using a historical dataset D.



Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

- **Approximate LP with Control Parameter.**
  - We approximate the original MILP by Equation (2) using the estimated $\hat{d}_{ij}$ and $\hat{g}_{ij}$.

$$
\begin{aligned}
\max \quad & \sum_{j \in Q} \sum_{i \in [M]} \alpha \hat{d}_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j} \hat{g}_{ij} x_{ij} \leq B_i, \ \forall i, \\
& \sum_{i} x_{ij} \leq 1, \ \forall j, \\
& x_{ij} \in \{0, 1\}, \ \forall i, j
\end{aligned}
\tag{2}
$$

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

- **Approximate LP with Control Parameter.**
  - We approximate the original MILP by Equation (2) using the estimated $\hat{d}_{ij}$ and $\hat{g}_{ij}$.
- **Routing via Learned Weights.**
  - We further relax Equation (2) to Equation (3)

$$
\begin{aligned}
\max \sum_{j \in Q} \sum_{i \in [M]} & \alpha \hat{d}_{ij} x_{ij} \\
\text{s.t.} \sum_{j} & \hat{g}_{ij} x_{ij} \leq B_i, \ \forall i, \\
\sum_{i} & x_{ij} \leq 1, \ \forall j, \\
& x_{ij} \in \{0,1\}, \ \forall i, j
\end{aligned}
\tag{2}
$$

$$
\begin{aligned}
\max \sum_{j \in Q} \sum_{i \in [M]} & \alpha \hat{d}_{ij} x_{ij} \\
\text{s.t.} \sum_{j} & \hat{g}_{ij} x_{ij} \leq B_i, \ \forall i, \\
\sum_{i} & x_{ij} \leq 1, \ \forall j, \\
& x_{ij} \in [0,1], \ \forall i, j
\end{aligned}
\tag{3}
$$

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

- **Approximate LP with Control Parameter.**
  - We approximate the original MILP by Equation (2) using the estimated $\hat{d}_{ij}$ and $\hat{g}_{ij}$.
- **Routing via Learned Weights.**
  - We further relax Equation (2) to Equation (3) and derive its dual in Equation (4).

$$\begin{aligned} \max \quad & \sum_{j \in Q} \sum_{i \in [M]} \alpha \hat{d}_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j} \hat{g}_{ij} x_{ij} \leq B_i, \ \forall i, \\ & \sum_{i} x_{ij} \leq 1, \ \forall j, \\ & x_{ij} \in \{0,1\}, \ \forall i, j \end{aligned} \quad (2)$$

$$\begin{aligned} \max \quad & \sum_{j \in Q} \sum_{i \in [M]} \alpha \hat{d}_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j} \hat{g}_{ij} x_{ij} \leq B_i, \ \forall i, \\ & \sum_{i} x_{ij} \leq 1, \ \forall j, \\ & x_{ij} \in [0,1], \ \forall i, j \end{aligned} \quad (3)$$

$$\begin{aligned} \min \quad & \sum_{i \in [M]} \gamma_i B_i + \sum_{j \in Q} \beta_j \\ \text{s.t.} \quad & \beta_j \geq \alpha \hat{d}_{ij} - \hat{g}_{ij} \gamma_i, \ \forall i, j, \\ & \gamma_i \geq 0, \quad \beta_j \geq 0, \ \forall i, j \end{aligned} \quad (4)$$

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

- By complementary slackness, we have:

$$x_{ij} > 0 \Leftrightarrow \beta_j = \max_i \left(\alpha \hat{d}_{ij} - \hat{g}_{ij}\gamma_i\right)$$

$$\begin{aligned} \min \quad & \sum_{i \in [M]} \gamma_i B_i + \sum_{j \in Q} \beta_j \\ \text{s.t.} \quad & \beta_j \geq \alpha \hat{d}_{ij} - \hat{g}_{ij}\gamma_i, \ \forall i, j, \\ & \gamma_i \geq 0, \quad \beta_j \geq 0, \ \forall i, j \end{aligned} \tag{4}$$

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

- By complementary slackness, we have:

$$x_{ij} > 0 \Leftrightarrow \beta_j = \max_i \left(\alpha \hat{d}_{ij} - \hat{g}_{ij}\gamma_i\right)$$

$$\begin{aligned} \min \quad & \sum_{i \in [M]} \gamma_i B_i + \sum_{j \in Q} \beta_j \\ \text{s.t.} \quad & \beta_j \geq \alpha \hat{d}_{ij} - \hat{g}_{ij}\gamma_i, \ \forall i, j, \\ & \gamma_i \geq 0, \quad \beta_j \geq 0, \ \forall i, j \end{aligned} \tag{4}$$

$$\min \sum_{i \in [M]} \gamma_i B_i + \sum_{j \in Q} \beta_j \longrightarrow F(\gamma) = \sum_i \gamma_i B_i + \sum_j \max_i \left(\alpha \hat{d}_{ij} - \hat{g}_{ij}\gamma_i\right) \ (5)$$

This motivates treating γ as a set of learnable weights across LLMs to aid the query routing.

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

$$F(\gamma) = \sum_i \gamma_i B_i + \sum_j \max_i \left(\alpha \hat{d}_{ij} - \hat{g}_{ij}\gamma_i\right) \ (5)$$

- Computing the global optimal γ for all Q is infeasible online

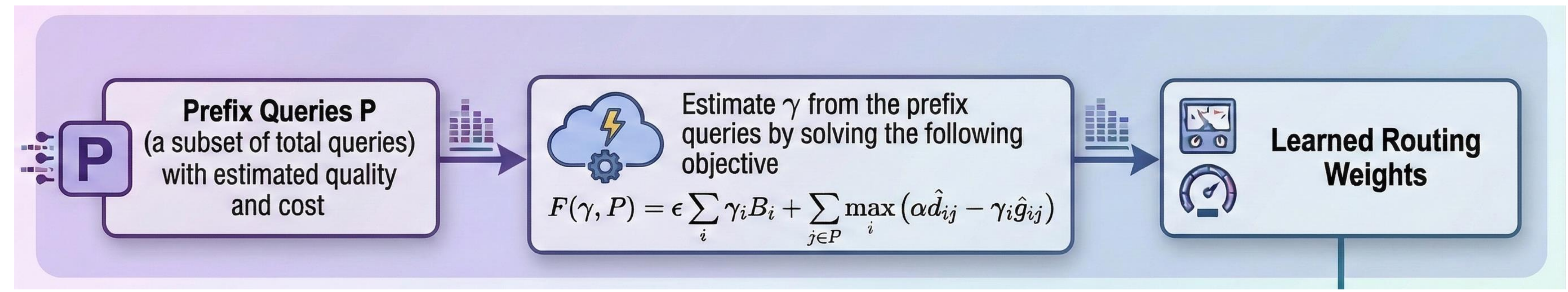
Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

$$F(\gamma) = \sum_i \gamma_i B_i + \sum_j \max_i \left(\alpha \hat{d}_{ij} - \hat{g}_{ij}\gamma_i\right) \text{ (5)}$$

- Computing the global optimal γ for all Q is infeasible online
- We estimate it with $\boldsymbol{\gamma}^*$ from the first $|P| = \epsilon|Q|$ queries by solving:

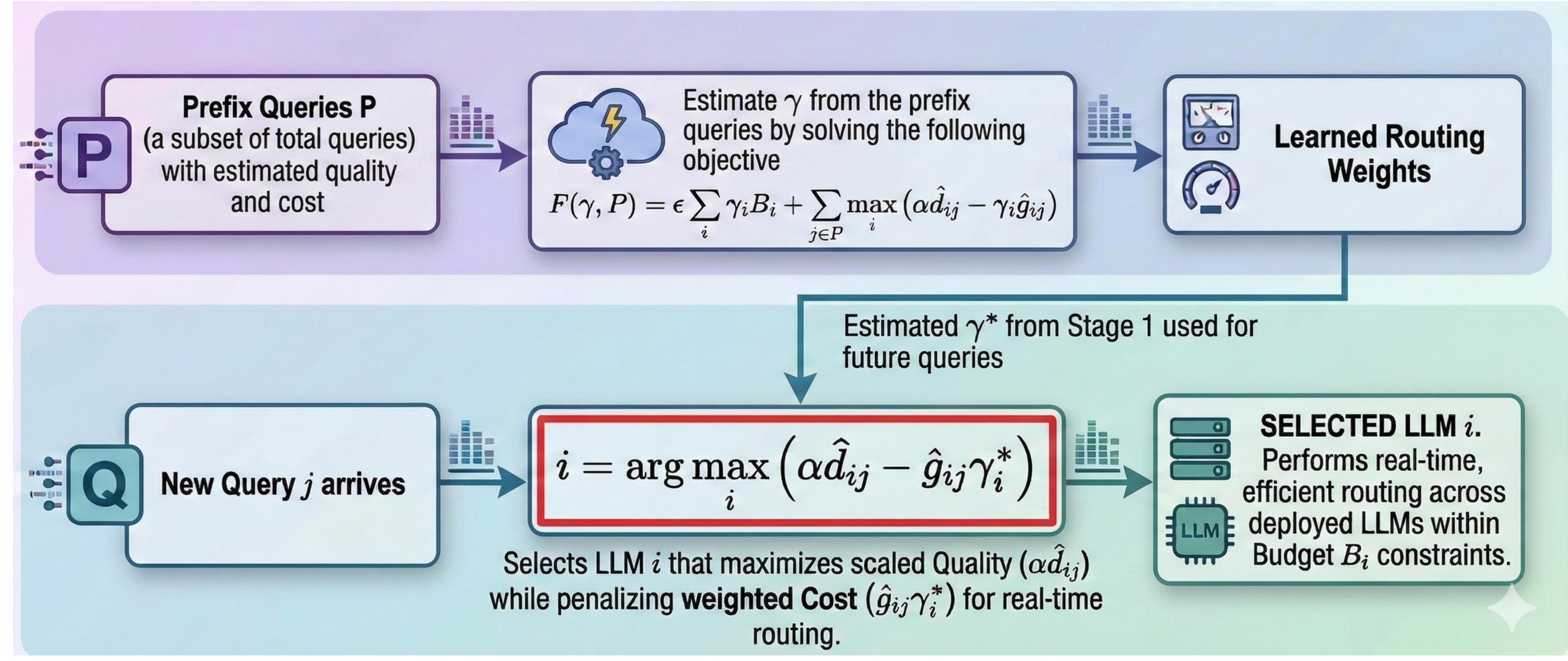
$$F(\gamma, P) = \epsilon \sum_i \gamma_i B_i + \sum_{j \in P} \max_i \left(\alpha \hat{d}_{ij} - \gamma_i \hat{g}_{ij}\right) \text{ (6)}$$



Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

- The estimated γ^* is then used to route future queries by assigning each query j to the LLM i that maximizes: $i = \arg\max_i(\alpha\hat{d}_{ij} - \hat{g}_{ij}\gamma_i^*)$



Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Online Routing from Observed Queries

- The estimated γ^* is then used to route future queries by assigning each query j to the LLM i that maximizes:

$$i = \arg\max_i(\alpha\hat{d}_{ij} - \hat{g}_{ij}\gamma_i^*)$$

- We provide formal theoretical guarantees demonstrating that our algorithm achieves a competitive ratio of $1 - o(1)$ under mild assumptions.

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Main Results

- As shown in the table below, our algorithm outperforms all baselines on average by **3.55×** in performance, **1.85×** in cost efficiency, and nearly **4.25×** in throughput.

Algorithm	RouterBench					SPROUT					Open LLM Leaderboard v2				
	Performance	Cost	Cost Efficiency	Throughput	Relative Performance	Performance	Cost	Cost Efficiency	Throughput	Relative Performance	Performance	Cost	Cost Efficiency	Throughput	Relative Performance
Random	1384.25	0.427	3243.25	3276	43.10%	2827.6	0.72	3927.29	4742	47.61%	953.0	0.741	1284.37	2877	49.89%
Greedy-Perf	1012.1	0.27	3742.379	1687	31.52%	764.9	0.406	1881.742	1083	12.88%	553.0	0.499	1107.91	1189	28.95%
Greedy-Cost	1626.25	0.46	3534.46	4061	50.64%	3934.7	0.849	4630.41	6789	66.25%	1051.0	0.766	1371.30	3164	55.02%
KNN-Perf	1005.1	0.27	3720.58	1677	31.3%	769.6	0.407	1888.46	1084	12.96%	556.0	0.498	1114.29	1194	29.11%
KNN-Cost	1592.05	0.46	3454.04	4027	49.58%	3905.1	0.85	4593.37	6709	65.75%	991.0	0.766	1293.07	3172	51.88%
BatchSplit	1838.05	0.458	4005.93	3903	57.24%	3975.5	0.83	4784.49	6221	66.94%	1059.0	0.76	1392.07	3099	55.44%
Roberta-Perf	154.5	0.077	2019.00	190	4.81%	458.9	0.283	1621.64	536	7.73%	153.0	0.207	738.21	283	8.01%
Roberta-Cost	481.4	0.129	3738.88	1292	14.99%	3996.2	0.848	4709.22	6765	67.29%	1044.0	0.766	1362.53	3173	54.66%
Ours	**2718.6**	0.447	**6075.58**	**5195**	**84.66%**	**4513.05**	0.815	**5536.74**	**7475**	**75.99%**	**1465.0**	0.711	**2060.3**	**3692**	**76.7%**
Offline Oracle (Algorithm Upper Bounds Reference)															
Approx Optimum($\hat{C}_{opt}$)	3211.35	0.46	6975.16	6225	100%	5938.99	0.85	6986.45	8781	100%	1910.0	0.765	2493.66	4319	100%
Optimum (C_{opt})	6376.9	0.46	13865.62	6436	198.57%	11934.4	0.848	14060.34	12336	200.94%	4688.0	0.763	6143.64	4688	245.44%

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Query Volume

- Vary query volume from 4000 to 12000 and our algorithm consistently outperforms all baselines, maintaining top performance and robustness as load increases.

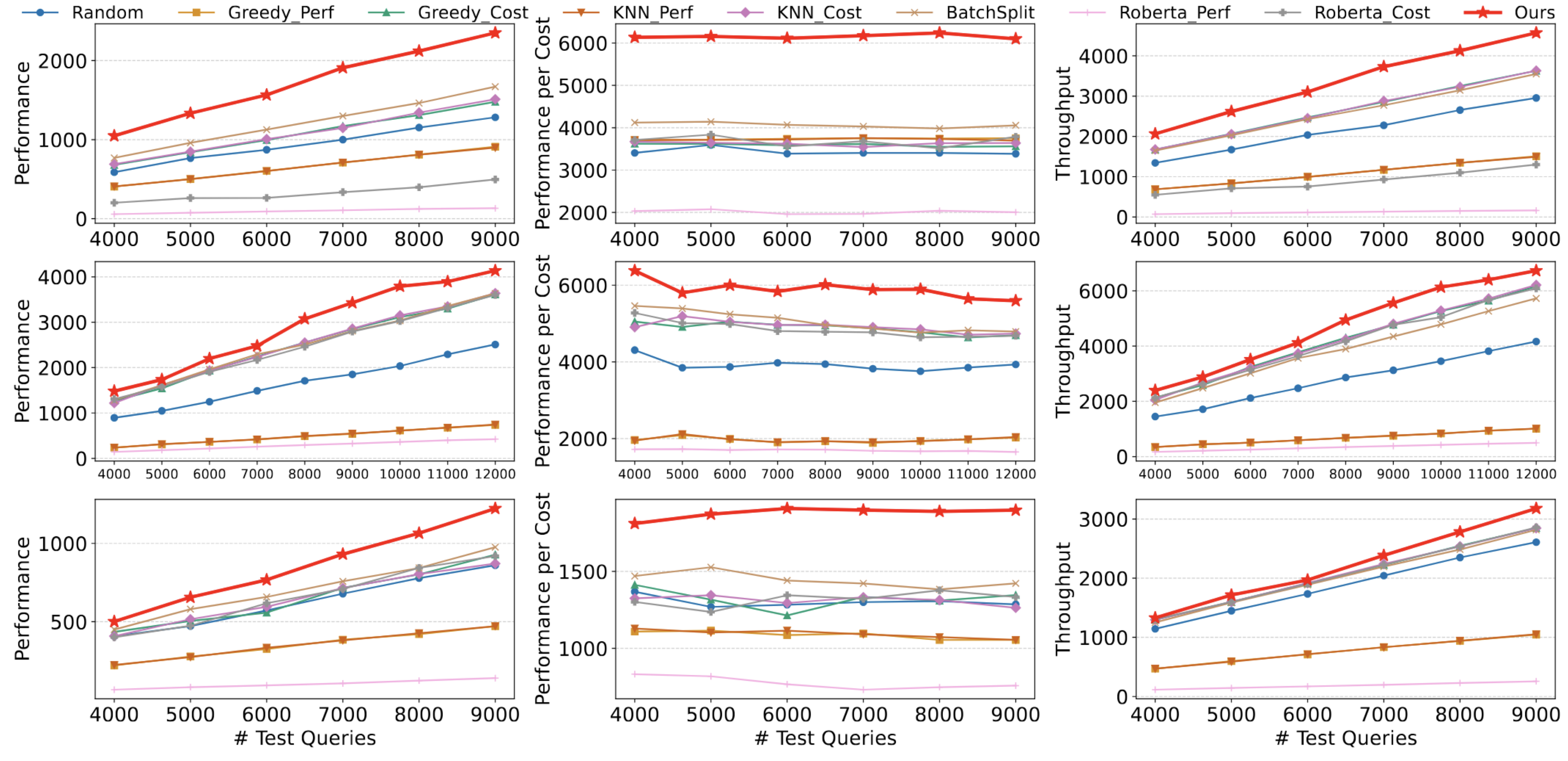
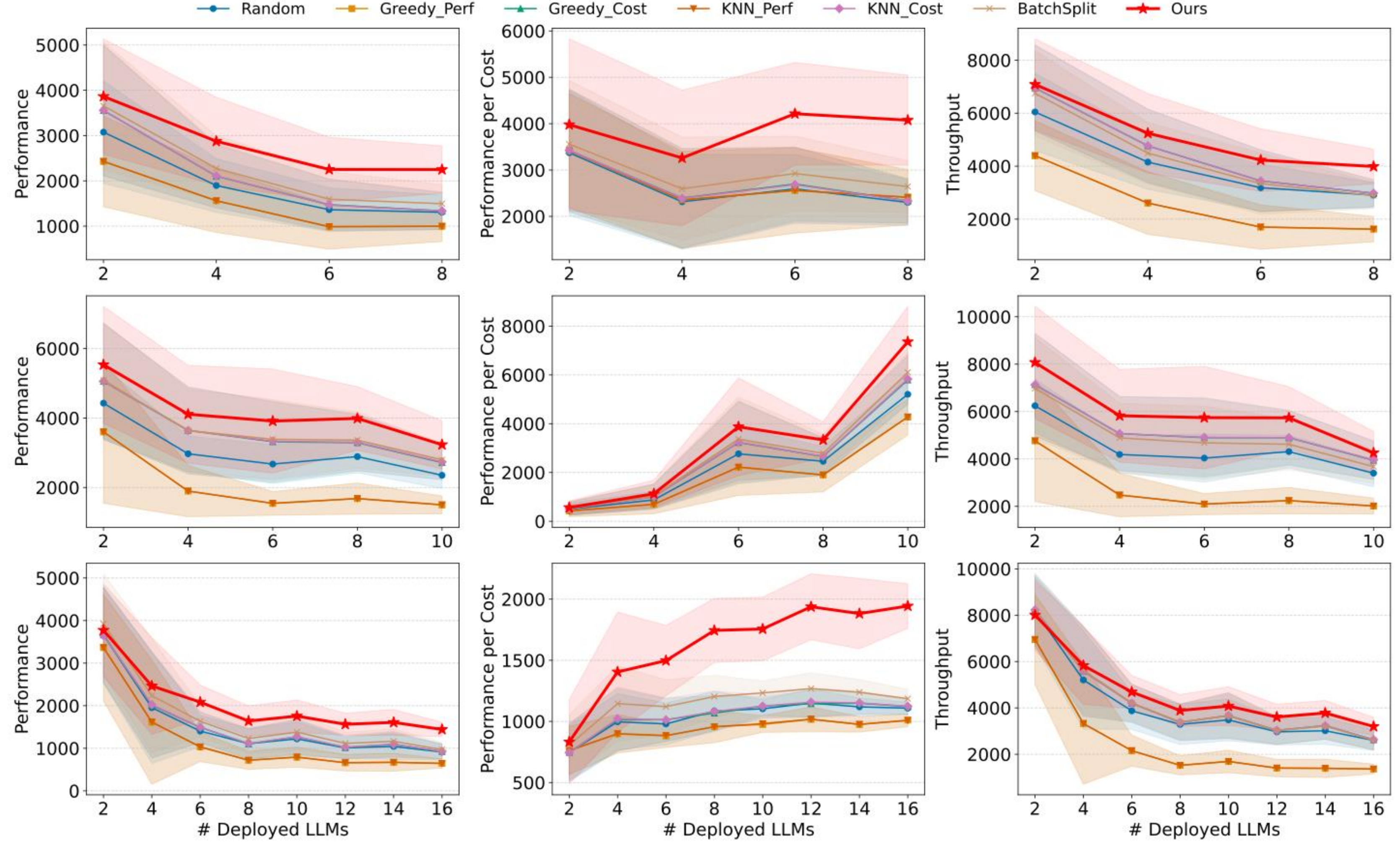


Figure 1: Results with test query volume varying from 4000 to 9000 (12000). Rows correspond to different datasets: RouterBench (top), SPROUT (middle), and Open LLM Leaderboard v2 (bottom).

Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Scalability Across LLM Deployments

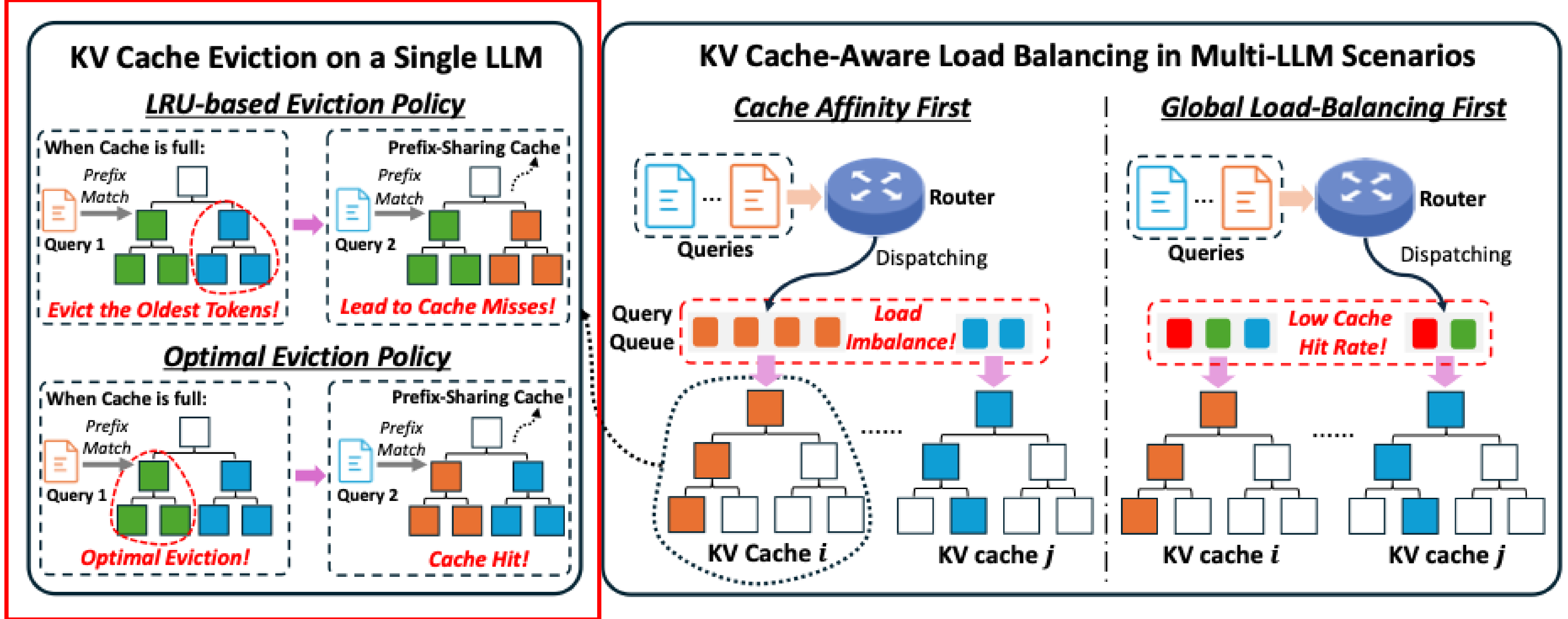
- Our method consistently outperforms training-free baselines under diverse LLM deployment configurations, highlighting its robustness.



Wu, F. and Silwal, S., Efficient Training-Free Online Routing for High-Volume Multi-LLM Serving. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Another Problem: KV Cache-Aware Load Balancing

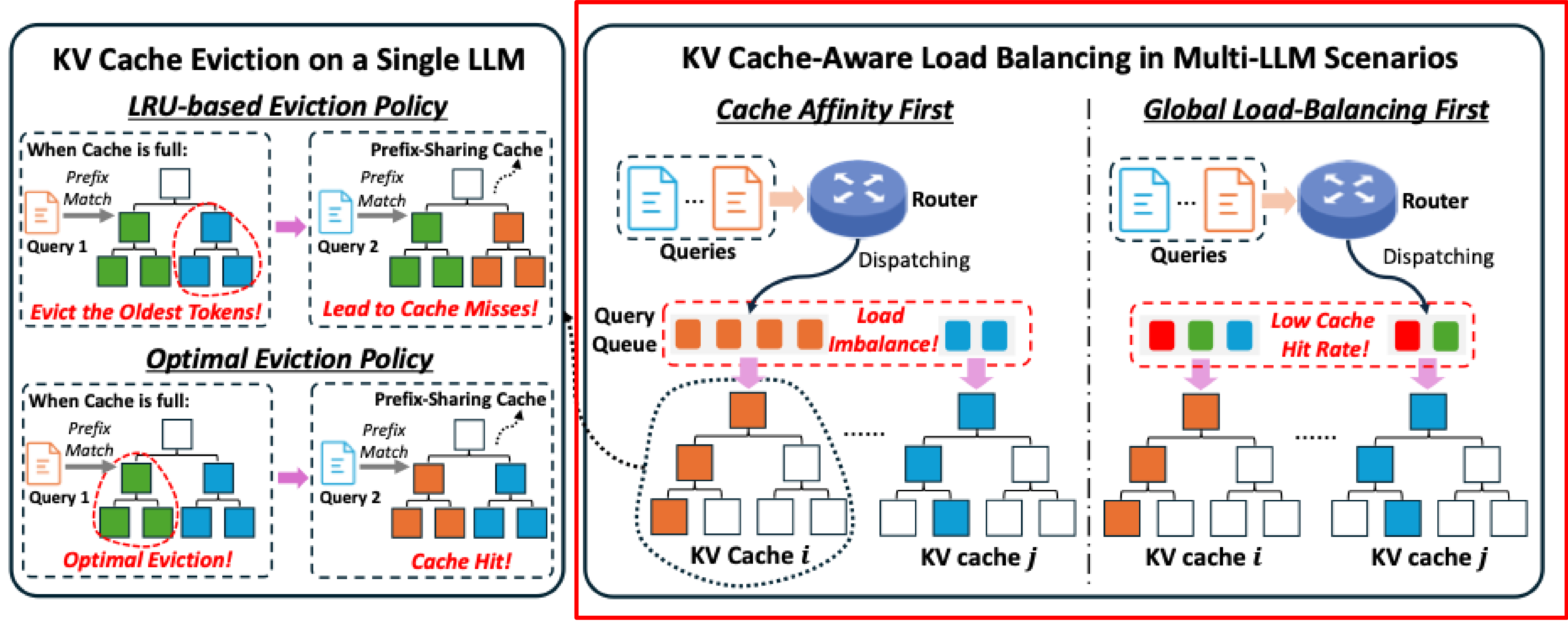
- When we further consider load balancing, routing becomes more complex…



Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Another Problem: KV Cache-Aware Load Balancing

- When we further consider load balancing, routing becomes more complex…



Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Current Heuristic Methods

- They adopt a static linear score that ranks LLMs based on cache hit rate and queue length
  - Dynamo (NVIDIA, https://github.com/ai-dynamo)
  - llm-d (llm-d, https://github.com/llm-d/llm-d)

Current Heuristic Methods

- They adopt a static linear score that ranks LLMs based on cache hit rate and queue length
  - Dynamo (NVIDIA, https://github.com/ai-dynamo)
  - llm-d (llm-d, https://github.com/llm-d/llm-d)
- Or employ a rule-based strategy that switches between highest-hit-rate and least-loaded routing using a predefined load-balance threshold
  - SGLang (https://github.com/sgl-project/sglang)

Current Heuristic Methods

- They adopt a static linear score that ranks LLMs based on cache hit rate and queue length
  - Dynamo (NVIDIA, https://github.com/ai-dynamo)
  - llm-d (llm-d, https://github.com/llm-d/llm-d)
- Or employ a rule-based strategy that switches between highest-hit-rate and least-loaded routing using a predefined load-balance threshold
  - SGLang (https://github.com/sgl-project/sglang)
- Limitations:
  - Inherently static and unable to adapt to dynamic query arrival patterns

Current Heuristic Methods

- They adopt a static linear score that ranks LLMs based on cache hit rate and queue length
  - Dynamo (NVIDIA, https://github.com/ai-dynamo)
  - llm-d (llm-d, https://github.com/llm-d/llm-d)
- Or employ a rule-based strategy that switches between highest-hit-rate and least-loaded routing using a predefined load-balance threshold
  - SGLang (https://github.com/sgl-project/sglang)
- Limitations:
  - Inherently static and unable to adapt to dynamic query arrival patterns.
  - The modeling of queue workload is limited to the **raw counts** of pending queries, which is overly simplistic.

Current Heuristic Methods

- They adopt a static linear score that ranks LLMs based on cache hit rate and queue length
  - Dynamo (NVIDIA, https://github.com/ai-dynamo)
  - llm-d (llm-d, https://github.com/llm-d/llm-d)
- Or employ a rule-based strategy that switches between highest-hit-rate and least-loaded routing using a predefined load-balance threshold
  - SGLang (https://github.com/sgl-project/sglang)
- Limitations:
  - Inherently static and unable to adapt to dynamic query arrival patterns.
  - The modeling of queue workload is limited to the **raw counts** of pending queries, which is overly simplistic.
  - Fundamentally limited in achieving optimal performance, as they are rooted in heuristics, lack formal modeling.

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Unified Model for KV Cache-Aware Load Balancing

- **Workers (LLMs):**
  - M workers (LLMs), indexed by $i \in [M]$.
  - Each worker m_i has a KV cache of size B_i tokens.

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Unified Model for KV Cache-Aware Load Balancing

- **Workers (LLMs):**
  - M workers (LLMs), indexed by $i \in [M]$.
  - Each worker m_i has a KV cache of size B_i tokens.
- **Queries:**
  - A sequence of N queries $Q = \{q_j\}_{j=1}^{N}$ arriving online.
  - Each query q_j has input length $|q_j|$and output length$|a_j|$.

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Unified Model for KV Cache-Aware Load Balancing

- **Workers (LLMs):**
  - M workers (LLMs), indexed by $i \in [M]$.
  - Each worker m_i has a KV cache of size B_i tokens.
- **Queries:**
  - A sequence of N queries $Q = \{q_j\}_{j=1}^{N}$ arriving online.
  - Each query q_j has input length $|q_j|$and output length$|a_j|$.
- **Worker state (after processing j queries):**
  - $P_i^{(j)}$:total accumulated query load on worker m_i.
  - $S_i^{(j)}$:cache state of worker m_i.
  - Initialization: $P_i^{(0)} = 0,\ S_i^{(0)} = \emptyset$ for all i.

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Unified Model for KV Cache-Aware Load Balancing

- **Service cost model:**
  - For query q_j on worker m_i, $h_{ij} = h\left(S_i^{(j-1)}, q_j\right)$ is the number of cache-hit tokens.
  - α_{cache} :per-token processing cost for cached tokens.
  - α_{miss} :per-token processing cost for uncached tokens.
  - O_{ij} :cost of generating the output a_j.
  - Total service cost $Cost_{ij}$ for m_i to process q_j :

$$Cost_{ij} = \alpha_{\text{CACHED}} \cdot h_{ij} + \alpha_{\text{MISS}} \cdot (|q_j| - h_{ij}) + O_{ij}$$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Unified Model for KV Cache-Aware Load Balancing

- **Service cost model:**
  - For query q_j on worker m_i, $h_{ij} = h\left(S_i^{(j-1)}, q_j\right)$ is the number of cache-hit tokens.
  - α_{cache} :per-token processing cost for cached tokens.
  - α_{miss} :per-token processing cost for uncached tokens.
  - O_{ij} :cost of generating the output a_j.
  - Total service cost $Cost_{ij}$ for m_i to process q_j :

$$Cost_{ij} = \alpha_{\text{CACHED}} \cdot h_{ij} + \alpha_{\text{MISS}} \cdot (|q_j| - h_{ij}) + O_{ij}$$

- **Routing decision:**
  - Binary variable $x_{ij} \in \{0, 1\}$ indicates whether query q_j is routed to worker m_i.
  - Constraint: $\sum_{i=1}^{M} x_{ij} = 1$ for all $j \in [N]$.

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Unified Model for KV Cache-Aware Load Balancing

- **State update after routing:** Once query q_j is assigned via x_{ij},
  - **Queue load:** the selected worker i accumulates the service cost:

$$P_i^{(j)} = P_i^{(j-1)} + x_{ij} \cdot Cost_{ij}.$$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Unified Model for KV Cache-Aware Load Balancing

- **State update after routing:** Once query q_j is assigned via x_{ij},
  - **Queue load:** the selected worker i accumulates the service cost:

$$P_i^{(j)} = P_i^{(j-1)} + x_{ij} \cdot Cost_{ij}.$$

  - **Cache update:** the selected worker updates its KV cache:

$$S_i^{(j)} = \begin{cases} \textsc{UpdateCache}(S_i^{(j-1)}, q_j, B_i) & \text{if } x_{ij} = 1 \\ S_i^{(j-1)} & \text{if } x_{ij} = 0 \end{cases}$$

where $UpdateCache(\cdot)$ is a cache policy (e.g., LRU).

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Unified Model for KV Cache-Aware Load Balancing

- **State update after routing:** Once query q_j is assigned via x_{ij},
  - **Queue load:** the selected worker i accumulates the service cost:
$$P_i^{(j)} = P_i^{(j-1)} + x_{ij} \cdot Cost_{ij}.$$
  - **Cache update:** the selected worker updates its KV cache:
$$S_i^{(j)} = \begin{cases} \textsc{UpdateCache}(S_i^{(j-1)}, q_j, B_i) & \text{if } x_{ij} = 1 \\ S_i^{(j-1)} & \text{if } x_{ij} = 0 \end{cases}$$

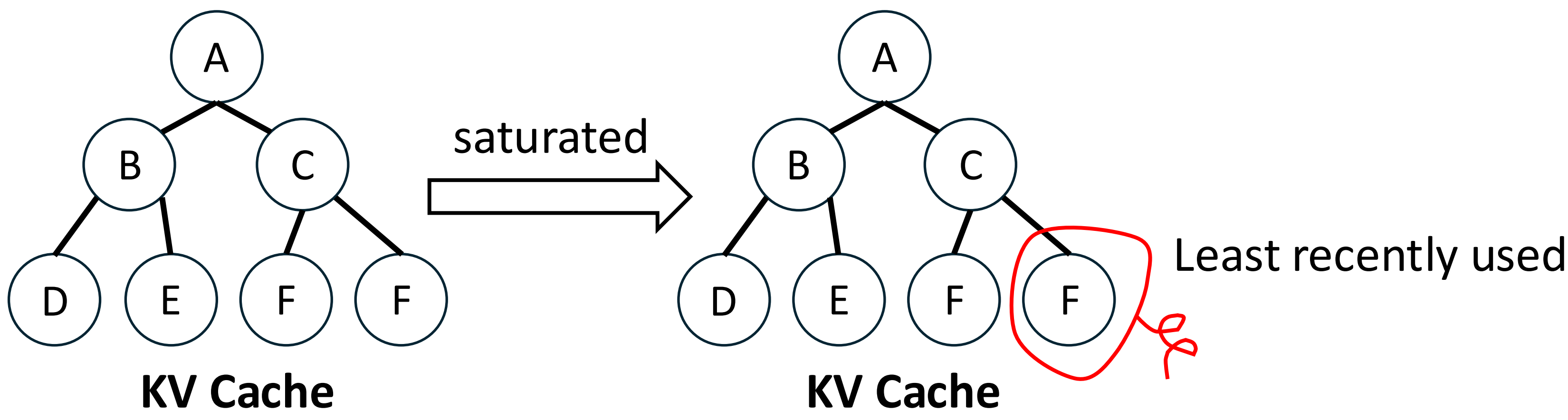
  where $UpdateCache(\cdot)$ is a cache policy (e.g., LRU).

- **Objective (makespan minimization):**
  - Choose assignments $x = \{x_{ij}\}$ to minimize the final maximum load:
$$\min_x \left(\max_{i \in [M]} \{P_i^{(N)}\} \right)$$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Theoretical Analysis of LRU-Based Eviction

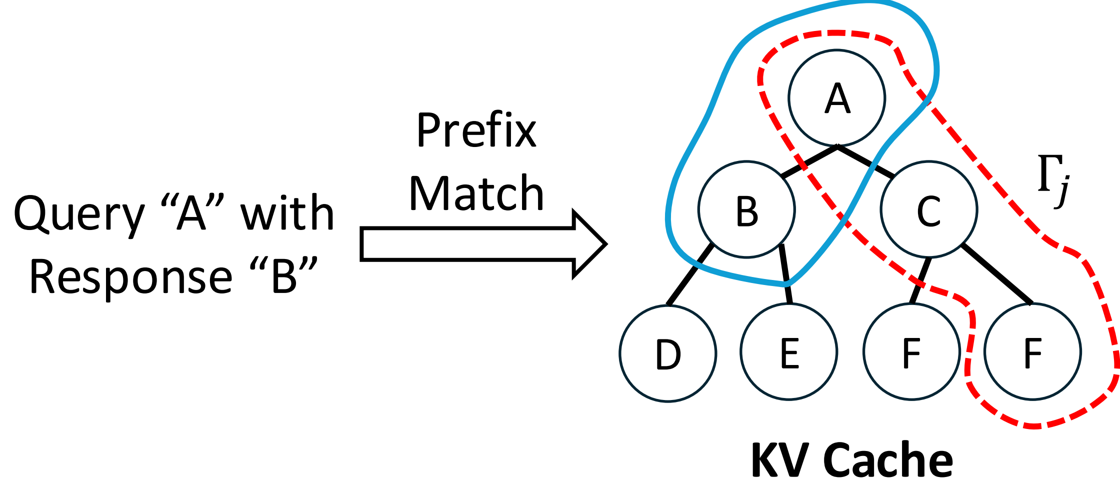
- **Leaf-LRU (L-LRU):**
  - We focus on **Leaf-LRU (L-LRU)**, the eviction algorithm used in **SGLang**.
  - SGLang stores KV values in a radix tree at **token granularity**.
  - When memory is saturated, L-LRU evicts the **least recently used leaf tokens**.



Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Theoretical Analysis of LRU-Based Eviction

- **Cache matching model:**
  - Define the **complete token path** $\Gamma_j := q_j \| a_j$ as the concatenation of the prompt and generated tokens.
  - Extend the query set to complete token paths: $\tilde{Q} := \{\Gamma_j\}_{j=1}^{N}$
  - Cache matching reduces to **longest-prefix matching** in the radix tree.



Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Theoretical Analysis of LRU-Based Eviction

- **Results**: L-LRU achieves an $O(n)$ competitive ratio in the worst case.
- Fixing KV cache size B_i, decreasing the minimum query length $\mathcal{L}$ drives the competitive ratio toward B_i.
- **Implication:** When query lengths are **highly imbalanced**, **L-LRU performance degrade**.

Theorem 5 (Batch). *Consider the continuous batch setting with batch size β. Let $\mathcal{L}_{max}$ and $\mathcal{L}$ denote the maximum and minimum lengths over all $\Gamma_j \in \widetilde{Q}$. If $\beta\mathcal{L}_{max} \leq B_i$, where B_i is the cache capacity of worker m_i, and all queries in a batch are distinct, then, the competitive ratio of* L-LRU *in RadixAttention is upper bounded by $(B_i - \mathcal{L} - \beta + 3)$ and lower bounded by $(B_i - \mathcal{L} - \beta + 2)$.*

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Randomized Leaf Token eviction (RLT)

- RLT marks accessed tokens by adding them to a marking set T during prefix matching.

Algorithm 1 RLT

1: Marking token set initialization: $\mathcal{T} \leftarrow \varnothing$
2: KV cache size: B_i
3: Complete token paths: $\widetilde{Q} \leftarrow Q$
4: **for** $\Gamma_j \in \widetilde{Q}$ **do**
5: $\quad S_i^{(j)} \leftarrow S_i^{(j-1)}$
6: $\quad$ **for** $t \in \Gamma_j$ **do**
7: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{t\}$
8: $\quad\quad$ **if** $|\mathcal{T}| = B_i + 1$ **then**
9: $\quad\quad\quad \mathcal{T} \leftarrow \{t\}$
10: $\quad\quad$ **if** $t \in S_i^{(j)}$ **then**
11: $\quad\quad\quad$ **continue**
12: $\quad\quad$ **else**
13: $\quad\quad\quad$ **if** $S_i^{(j)}$ is full **then**
14: $\quad\quad\quad\quad U \leftarrow \{\text{leaf tokens in } S_i^{(j)}\} \setminus \mathcal{T}$
15: $\quad\quad\quad\quad$ Choose $u \in U$ uniformly at random
16: $\quad\quad\quad\quad S_i^{(j)} \leftarrow \textsc{Evict}(S_i^{(j)}, u)$
17: $\quad\quad\quad S_i^{(j)} \leftarrow \textsc{Load}(S_i^{(j)}, t)$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Randomized **L**eaf **T**oken eviction (RLT)

- RLT marks accessed tokens by adding them to a marking set T during prefix matching.

- Once $B_i + 1$ unique tokens have been marked, RLT clears all marks except the most recently accessed token.

Algorithm 1 RLT

1: Marking token set initialization: $\mathcal{T} \leftarrow \varnothing$
2: KV cache size: B_i
3: Complete token paths: $\widetilde{Q} \leftarrow Q$
4: **for** $\Gamma_j \in \widetilde{Q}$ **do**
5: $\quad S_i^{(j)} \leftarrow S_i^{(j-1)}$
6: $\quad$ **for** $t \in \Gamma_j$ **do**
7: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{t\}$
8: $\quad\quad$ **if** $|\mathcal{T}| = B_i + 1$ **then**
9: $\quad\quad\quad \mathcal{T} \leftarrow \{t\}$
10: $\quad\quad$ **if** $t \in S_i^{(j)}$ **then**
11: $\quad\quad\quad$ **continue**
12: $\quad\quad$ **else**
13: $\quad\quad\quad$ **if** $S_i^{(j)}$ is full **then**
14: $\quad\quad\quad\quad U \leftarrow \{\text{leaf tokens in } S_i^{(j)}\} \setminus \mathcal{T}$
15: $\quad\quad\quad\quad$ Choose $u \in U$ uniformly at random
16: $\quad\quad\quad\quad S_i^{(j)} \leftarrow \textsc{Evict}(S_i^{(j)}, u)$
17: $\quad\quad\quad S_i^{(j)} \leftarrow \textsc{Load}(S_i^{(j)}, t)$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Randomized **L**eaf **T**oken eviction (RLT)

- RLT marks accessed tokens by adding them to a marking set T during prefix matching.
- Once $B_i + 1$ unique tokens have been marked, RLT clears all marks except the most recently accessed token.
- When saturated, RLT **randomly evicts an unmarked leaf token**.

Algorithm 1 RLT

1: Marking token set initialization: $\mathcal{T} \leftarrow \varnothing$
2: KV cache size: B_i
3: Complete token paths: $\widetilde{Q} \leftarrow Q$
4: **for** $\Gamma_j \in \widetilde{Q}$ **do**
5: $\quad S_i^{(j)} \leftarrow S_i^{(j-1)}$
6: $\quad$ **for** $t \in \Gamma_j$ **do**
7: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{t\}$
8: $\quad\quad$ **if** $|\mathcal{T}| = B_i + 1$ **then**
9: $\quad\quad\quad \mathcal{T} \leftarrow \{t\}$
10: $\quad\quad$ **if** $t \in S_i^{(j)}$ **then**
11: $\quad\quad\quad$ **continue**
12: $\quad\quad$ **else**
13: $\quad\quad\quad$ **if** $S_i^{(j)}$ is full **then**
14: $\quad\quad\quad\quad U \leftarrow \{\text{leaf tokens in } S_i^{(j)}\} \setminus \mathcal{T}$
15: $\quad\quad\quad\quad$ Choose $u \in U$ uniformly at random
16: $\quad\quad\quad\quad S_i^{(j)} \leftarrow \textsc{Evict}(S_i^{(j)}, u)$
17: $\quad\quad\quad S_i^{(j)} \leftarrow \textsc{Load}(S_i^{(j)}, t)$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Randomized **L**eaf **T**oken eviction (RLT)

- RLT achieves a $O(\log n)$ competitive ratio, a logarithmic improvement over L-LRU.

Corollary 7 (Batch). *RLT is $\Theta(\log(B_i - \mathcal{L} - \beta))$-competitive on worker m_i with capacity B_i under continuous batching setting, where $\mathcal{L}$ is the minimal length over all $\Gamma_j \in \widetilde{Q}$, and β is the batchsize.*

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Randomized **L**eaf **T**oken eviction (RLT)

- RLT achieves a $O(\log n)$ competitive ratio, a logarithmic improvement over L-LRU.

Corollary 7 (Batch). *RLT is $\Theta(\log(B_i - \mathcal{L} - \beta))$-competitive on worker m_i with capacity B_i under continuous batching setting, where $\mathcal{L}$ is the minimal length over all $\Gamma_j \in \widetilde{Q}$, and β is the batchsize.*

- No dependent algorithm can achieve a better competitive ratio than RLT.

Corollary 9 (Batch). *No randomized eviction algorithm can achieve a competitive ratio better than $\Theta(\log(B_i - \mathcal{L} - \beta))$ on worker m_i with cache capability of B_i in the continuous batching setting, where $\mathcal{L}$ denotes the minimal length over all $\Gamma_j \in \widetilde{Q}$ and β is the batch size.*

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Learning-Based Greedy Routing

- End-to-end latency for worker m_i processing q_j:

$$E_{ij} = Cost_{ij} + P_i^{(j-1)}$$

Algorithm 2 LBGR

```
1: ▷ /* Online Routing Thread */
2: for q_j ∈ Q do
3:     for i ∈ [M] do
4:         Estimate the h̃_ij via global radix tree
5:         Ĉost_ij ← α_CACHED h̃_ij + α_MISS (|q_j| − h̃_ij)
6:         φ_ij ← φ(h̃_ij, |q_j| − h̃_ij, P̃_i^(j−1))
7:         Ê_ij ← Ĉost_ij + P̃_i^(j−1) + θ_i^⊤ φ_ij
8:     i* ← arg min_{i∈[M]} Ê_ij, x_ij := 1[i = i*]
9:     ∀i ∈ [M], P̃_i^(j) ← P̃_i^(j−1) + x_ij Ĉost_ij
10:    Route q_j to m_{i*}
11: ▷ /* Online Updating Thread */
12: if observe actual latency E_ij then
13:    θ_i ← OnlineUpdate(θ_i, φ_ij, E_ij − Ê_ij)
14:    P̃_i ← ReleaseLoad(P̃_i, Ĉost_ij, ρ, Δt)
15: ▷ /* Background Decay Thread */
16: Every Δt time units: ∀i, P̃_i ← ρP̃_i
```

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Learning-Based Greedy Routing

- End-to-end latency for worker m_i processing q_j:

$$E_{ij} = Cost_{ij} + P_i^{(j-1)}$$

- **LBGR predicts latency via**:

$$\widehat{E}_{ij} = \underbrace{\widehat{Cost}_{ij}}_{\text{service time estimation}} + \underbrace{\widetilde{P}_i^{(j-1)}}_{\text{queue load estimation}} + \underbrace{\theta_i^\top \phi_{ij}}_{\text{residual correction}}$$

Algorithm 2 LBGR

1: ▷ /* Online Routing Thread */
2: **for** $q_j \in Q$ **do**
3: **for** $i \in [M]$ **do**
4: Estimate the $\widetilde{h}_{ij}$ via global radix tree
5: $\widehat{Cost}_{ij} \leftarrow \alpha_{\text{CACHED}} \widetilde{h}_{ij} + \alpha_{\text{MISS}} \left(|q_j| - \widetilde{h}_{ij} \right)$
6: $\phi_{ij} \leftarrow \phi\left(\widetilde{h}_{ij},\, |q_j| - \widetilde{h}_{ij},\, \widetilde{P}_i^{(j-1)} \right)$
7: $\widehat{E}_{ij} \leftarrow \widehat{Cost}_{ij} + \widetilde{P}_i^{(j-1)} + \theta_i^\top \phi_{ij}$
8: $i^* \leftarrow \arg\min_{i \in [M]} \widehat{E}_{ij},\ x_{ij} := \mathbf{1}[i = i^*]$
9: $\forall i \in [M],\ \widetilde{P}_i^{(j)} \leftarrow \widetilde{P}_i^{(j-1)} + x_{ij}\, \widehat{Cost}_{ij}$
10: Route q_j to m_{i^*}
11: ▷ /* Online Updating Thread */
12: **if** observe actual latency E_{ij} **then**
13: $\theta_i \leftarrow \text{OnlineUpdate}\left(\theta_i, \phi_{ij}, E_{ij} - \widehat{E}_{ij} \right)$
14: $\widetilde{P}_i \leftarrow \text{ReleaseLoad}\left(\widetilde{P}_i, \widehat{Cost}_{ij}, \rho, \Delta t \right)$
15: ▷ /* Background Decay Thread */
16: Every Δt time units: $\forall i,\ \widetilde{P}_i \leftarrow \rho \widetilde{P}_i$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Learning-Based Greedy Routing

- End-to-end latency for worker m_i processing q_j:

$$E_{ij} = Cost_{ij} + P_i^{(j-1)}$$

- **LBGR predicts latency via**:

$$\widehat{E}_{ij} = \underbrace{\widehat{Cost}_{ij}}_{\text{service time estimation}} + \underbrace{\widetilde{P}_i^{(j-1)}}_{\text{queue load estimation}} + \underbrace{\theta_i^\top \phi_{ij}}_{\text{residual correction}}$$

- **Service time estimation**:

$$\widehat{Cost}_{ij} = \alpha_{\text{CACHED}} \cdot \widetilde{h}_{ij} + \alpha_{\text{MISS}} \cdot (|q_j| - \widetilde{h}_{ij})$$

Algorithm 2 LBGR

1: ▷ `/* Online Routing Thread */`
2: **for** $q_j \in Q$ **do**
3: **for** $i \in [M]$ **do**
4: Estimate the $\widetilde{h}_{ij}$ via global radix tree
5: $\widehat{Cost}_{ij} \leftarrow \alpha_{\text{CACHED}} \widetilde{h}_{ij} + \alpha_{\text{MISS}} (|q_j| - \widetilde{h}_{ij})$
6: $\phi_{ij} \leftarrow \phi\big(\widetilde{h}_{ij},\, |q_j| - \widetilde{h}_{ij},\, \widetilde{P}_i^{(j-1)}\big)$
7: $\widehat{E}_{ij} \leftarrow \widehat{Cost}_{ij} + \widetilde{P}_i^{(j-1)} + \theta_i^\top \phi_{ij}$
8: $i^* \leftarrow \arg\min_{i \in [M]} \widehat{E}_{ij}$, $x_{ij} := \mathbf{1}[i = i^*]$
9: $\forall i \in [M]$, $\widetilde{P}_i^{(j)} \leftarrow \widetilde{P}_i^{(j-1)} + x_{ij} \widehat{Cost}_{ij}$
10: Route q_j to m_{i^*}
11: ▷ `/* Online Updating Thread */`
12: **if** observe actual latency E_{ij} **then**
13: $\theta_i \leftarrow \textsc{OnlineUpdate}\big(\theta_i, \phi_{ij}, E_{ij} - \widehat{E}_{ij}\big)$
14: $\widetilde{P}_i \leftarrow \textsc{ReleaseLoad}\big(\widetilde{P}_i, \widehat{Cost}_{ij}, \rho, \Delta t\big)$
15: ▷ `/* Background Decay Thread */`
16: Every Δt time units: $\forall i$, $\widetilde{P}_i \leftarrow \rho \widetilde{P}_i$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Learning-Based Greedy Routing

- End-to-end latency for worker m_i processing q_j:

$$E_{ij} = Cost_{ij} + P_i^{(j-1)}$$

- **LBGR predicts latency via**:

$$\widehat{E}_{ij} = \underbrace{\widehat{Cost}_{ij}}_{\text{service time estimation}} + \underbrace{\widetilde{P}_i^{(j-1)}}_{\text{queue load estimation}} + \underbrace{\theta_i^\top \phi_{ij}}_{\text{residual correction}}$$

- **Service time estimation**:

$$\widehat{Cost}_{ij} = \alpha_{\text{CACHED}} \cdot \widetilde{h}_{ij} + \alpha_{\text{MISS}} \cdot (|q_j| - \widetilde{h}_{ij})$$

- **Queue load estimation**:

$$\widetilde{P}_i^{(j)} = \widetilde{P}_i^{(j-1)} + x_{ij} \cdot \widehat{Cost}_{ij}.$$

Algorithm 2 LBGR

1: ▷ `/* Online Routing Thread */`
2: **for** $q_j \in Q$ **do**
3: **for** $i \in [M]$ **do**
4: Estimate the $\widetilde{h}_{ij}$ via global radix tree
5: $\widehat{Cost}_{ij} \leftarrow \alpha_{\text{CACHED}} \widetilde{h}_{ij} + \alpha_{\text{MISS}} (|q_j| - \widetilde{h}_{ij})$
6: $\phi_{ij} \leftarrow \phi(\widetilde{h}_{ij}, |q_j| - \widetilde{h}_{ij}, \widetilde{P}_i^{(j-1)})$
7: $\widehat{E}_{ij} \leftarrow \widehat{Cost}_{ij} + \widetilde{P}_i^{(j-1)} + \theta_i^\top \phi_{ij}$
8: $i^* \leftarrow \arg\min_{i \in [M]} \widehat{E}_{ij}, x_{ij} := \mathbf{1}[i = i^*]$
9: $\forall i \in [M], \widetilde{P}_i^{(j)} \leftarrow \widetilde{P}_i^{(j-1)} + x_{ij} \widehat{Cost}_{ij}$
10: Route q_j to m_{i^*}
11: ▷ `/* Online Updating Thread */`
12: **if** observe actual latency E_{ij} **then**
13: $\theta_i \leftarrow \text{ONLINEUPDATE}(\theta_i, \phi_{ij}, E_{ij} - \widehat{E}_{ij})$
14: $\widetilde{P}_i \leftarrow \text{RELEASELOAD}(\widetilde{P}_i, \widehat{Cost}_{ij}, \rho, \Delta t)$
15: ▷ `/* Background Decay Thread */`
16: Every Δt time units: $\forall i, \widetilde{P}_i \leftarrow \rho \widetilde{P}_i$

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Learning-Based Greedy Routing

- End-to-end latency for worker m_i processing q_j:

$$E_{ij} = Cost_{ij} + P_i^{(j-1)}$$

- **LBGR predicts latency via**:

$$\widehat{E}_{ij} = \underbrace{\widehat{Cost}_{ij}}_{\text{service time estimation}} + \underbrace{\widetilde{P}_i^{(j-1)}}_{\text{queue load estimation}} + \underbrace{\theta_i^\top \phi_{ij}}_{\text{residual correction}}$$

- **Service time estimation**:

$$\widehat{Cost}_{ij} = \alpha_{\text{CACHED}} \cdot \widetilde{h}_{ij} + \alpha_{\text{MISS}} \cdot (|q_j| - \widetilde{h}_{ij})$$

- **Queue load estimation**:

$$\widetilde{P}_i^{(j)} = \widetilde{P}_i^{(j-1)} + x_{ij} \cdot \widehat{Cost}_{ij}.$$

- **Residual model:** For each m_i, learn an **online linear regression** with parameters $\boldsymbol{\theta}_i$ to correct residual latency bias. For each q_j, extract feature

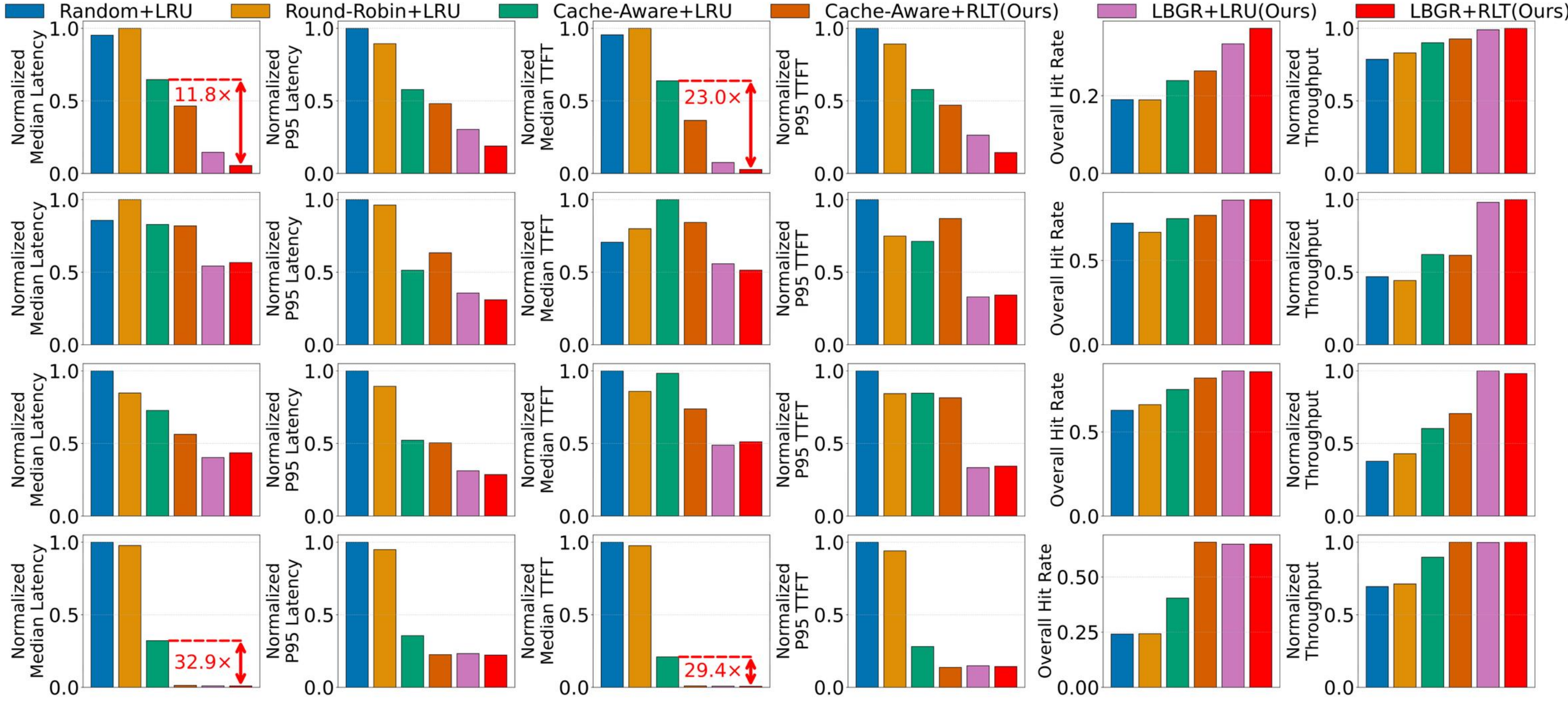
$$\phi_{ij} := \phi\big(\widetilde{h}_{ij},\, |q_j| - \widetilde{h}_{ij},\, \widetilde{P}_i^{(j-1)}\big)$$

Algorithm 2 LBGR

1: ▷ /* Online Routing Thread */
2: **for** $q_j \in Q$ **do**
3:   **for** $i \in [M]$ **do**
4:     Estimate the $\widetilde{h}_{ij}$ via global radix tree
5:     $\widehat{Cost}_{ij} \leftarrow \alpha_{\text{CACHED}} \widetilde{h}_{ij} + \alpha_{\text{MISS}} (|q_j| - \widetilde{h}_{ij})$
6:     $\phi_{ij} \leftarrow \phi\big(\widetilde{h}_{ij},\, |q_j| - \widetilde{h}_{ij},\, \widetilde{P}_i^{(j-1)}\big)$
7:     $\widehat{E}_{ij} \leftarrow \widehat{Cost}_{ij} + \widetilde{P}_i^{(j-1)} + \theta_i^\top \phi_{ij}$
8:   $i^* \leftarrow \arg\min_{i \in [M]} \widehat{E}_{ij}$, $x_{ij} := \mathbf{1}[i = i^*]$
9:   $\forall i \in [M]$, $\widetilde{P}_i^{(j)} \leftarrow \widetilde{P}_i^{(j-1)} + x_{ij} \widehat{Cost}_{ij}$
10:   Route q_j to m_{i^*}
11: ▷ /* Online Updating Thread */
12: **if** observe actual latency E_{ij} **then**
13:   $\theta_i \leftarrow \text{ONLINEUPDATE}\big(\theta_i, \phi_{ij}, E_{ij} - \widehat{E}_{ij}\big)$
14:   $\widetilde{P}_i \leftarrow \text{RELEASELOAD}\big(\widetilde{P}_i, \widehat{Cost}_{ij}, \rho, \Delta t\big)$
15: ▷ /* Background Decay Thread */
16: Every Δt time units: $\forall i$, $\widetilde{P}_i \leftarrow \rho \widetilde{P}_i$

Main Results

- Extensive experiments demonstrate improvements of up to **6.92× in cache hit rate,11.96× reduction in latency, 14.06× reduction in time-to-first-token (TTFT), and 77.4% increase in throughput** compared with the **state-of-the–art baseline, SGLang.**



Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Main Results

- Introduce negligible overhead compared with baselines.

Table 2: Ablation comparison of performance and runtime overhead between Cache-Aware+LRU and our methods on the GSP benchmark. Time-based metrics (↓) are reported in milliseconds (ms), while hit rate (↑) and throughput (↑) are measured in percentage and requests per second, respectively.

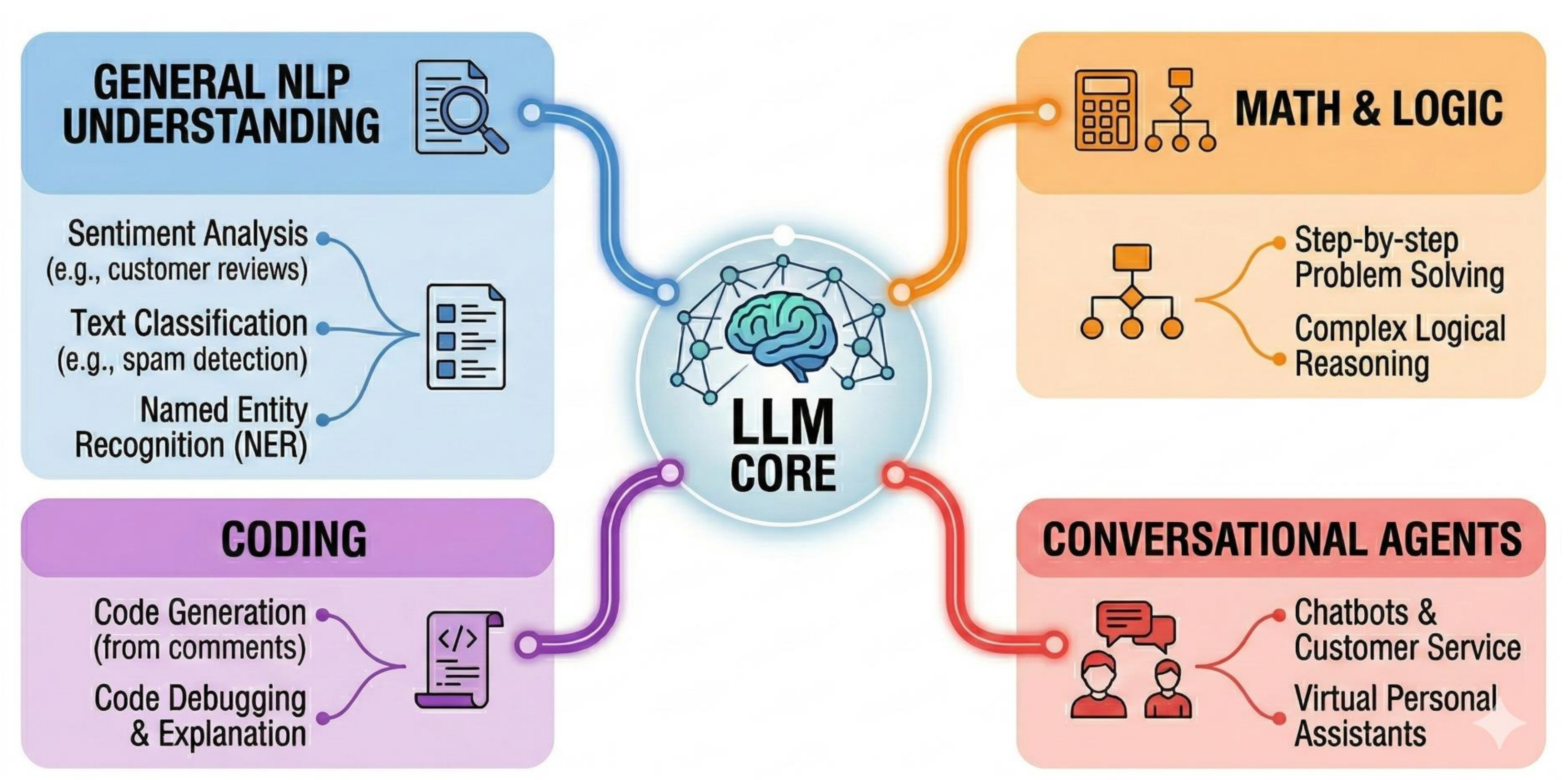
Method	P50 Latency	P95 Latency	P50 TTFT	P95 TTFT	Hit Rate	Throughput	Average Eviction Time	Average Routing Time
Cache-Aware+LRU	26680.55	46766.77	25022.76	46139.36	23.89%	10.73	0.13	**0.47**
Cache-Aware+RLT (Ours)	19191.25	38917.27	14332.81	37504.69	26.36%	11.05	0.71	0.51
LBGR+LRU (Ours)	6025.11	24561.47	2958.01	21073.78	33.33%	11.80	**0.09**	1.03
LBGR+RLT (Ours)	**2263.61**	**15334.89**	**1088.57**	**11495.05**	**37.31%**	**11.92**	1.05	1.45

Wu, F., Silwal, S. and Zhang, Q., Randomization Boosts KV Caching, Learning Balances Query Load: A Joint Perspective, in ICLR 2026

Large Language Model with Diverse Capabilities

- One fundamental question for LLM-based applications:

How can we accurately capture and assess their strengths?



Large Language Model with Diverse Capabilities

- One fundamental question for LLM-based applications:

How can we accurately capture and assess their strengths?

- One widely adopted method: Evaluating them on query benchmarks!

Evaluating LLM Capabilities via Queries

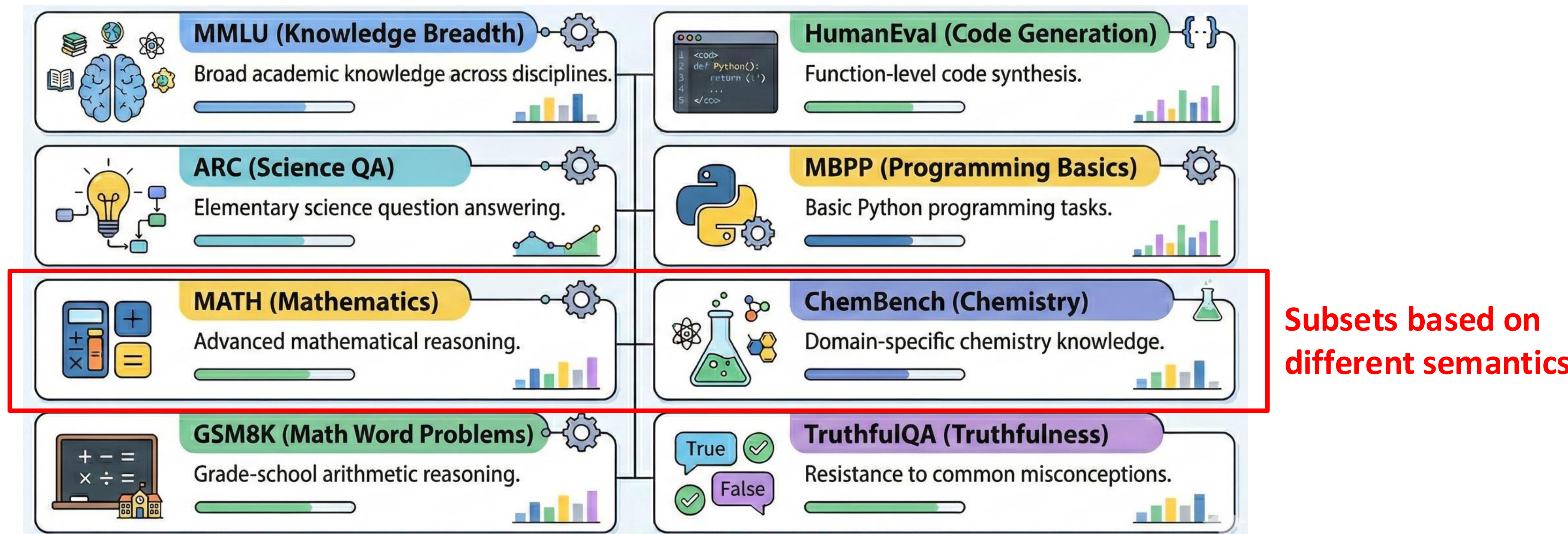
- Queries implicitly specify diverse capability requirements.

Evaluating LLM Capabilities via Queries

- Queries implicitly specify diverse capability requirements.
- LLM performance (capability) is inherently query-dependent.

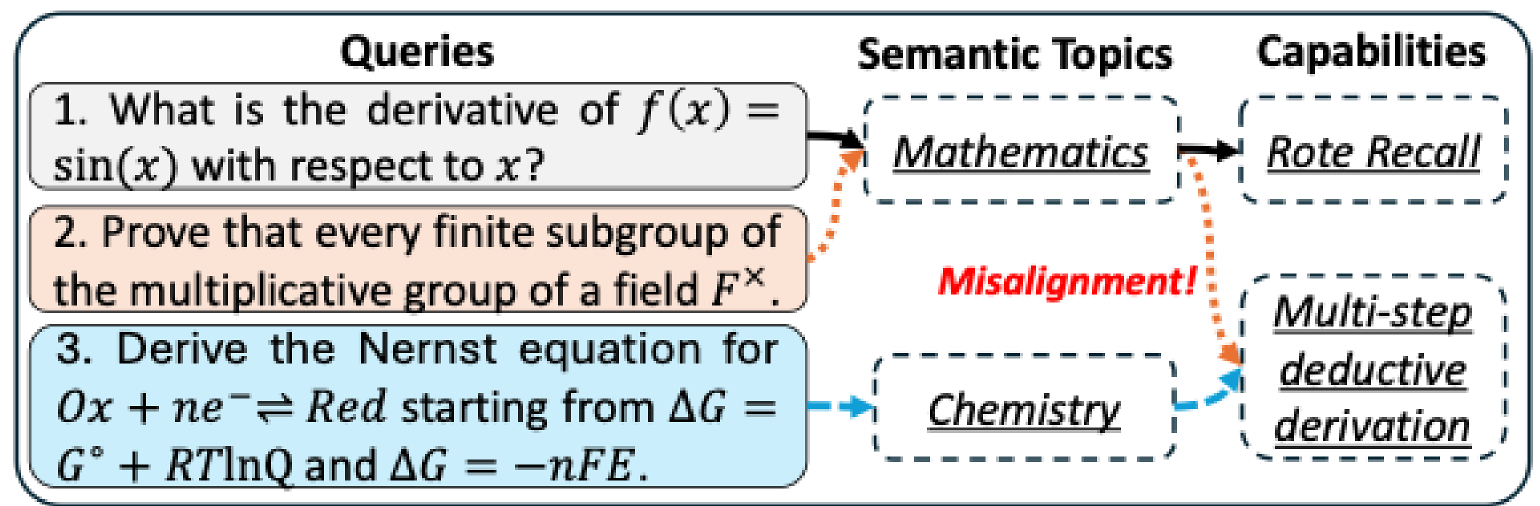
Evaluating LLM Capabilities via Queries

- Queries implicitly specify diverse capability requirements.
- LLM performance (capability) is inherently query-dependent.
- **Common practice: human semantic labels → capability proxies → subset evaluation**



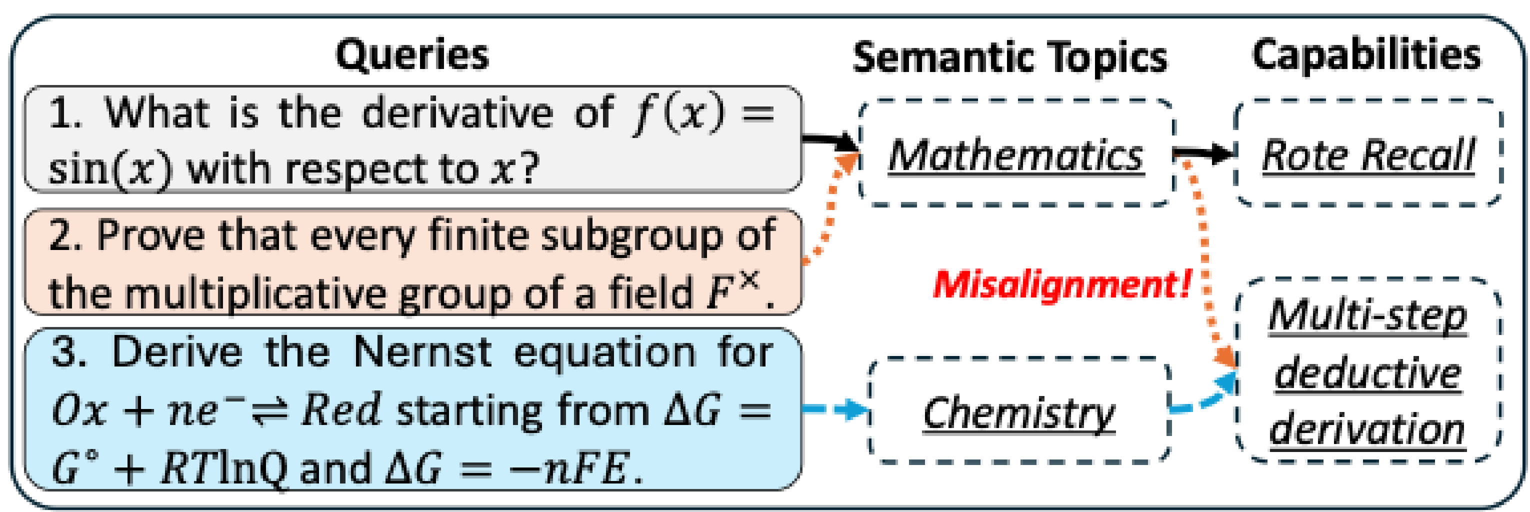
Semantics Often Fail to Reflect Underlying Capabilities!

- A single "Mathematics" label can include queries requiring vastly different levels of capabilities, or even a combination of distinct capability requirements.



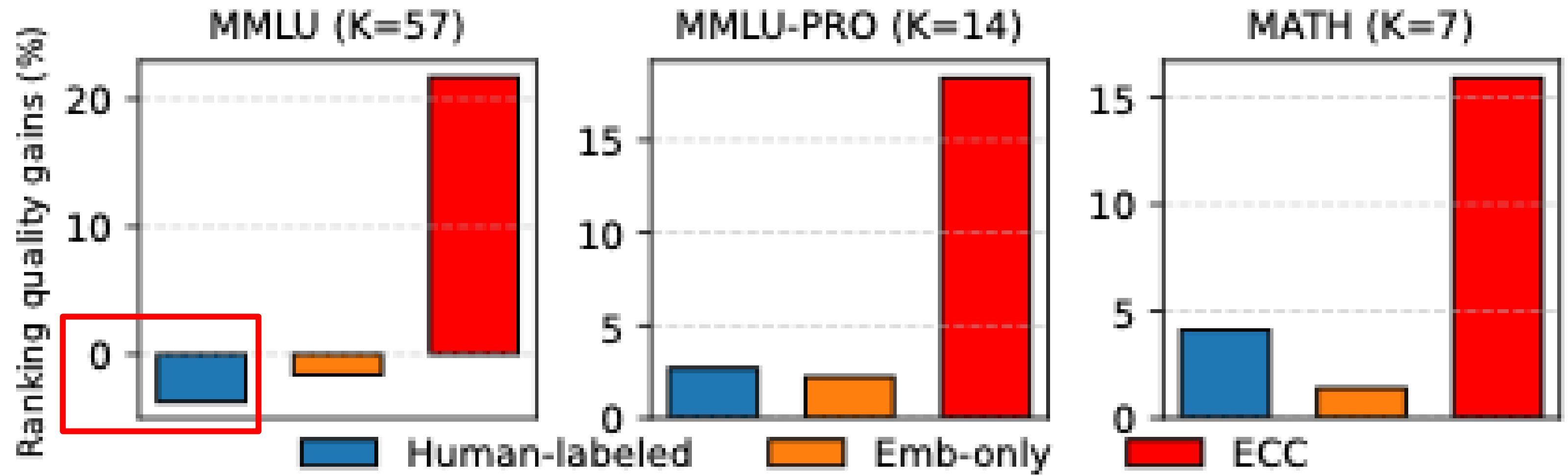
Semantics Often Fail to Reflect Underlying Capabilities!

- A single “Mathematics” label can include queries requiring vastly different levels of capabilities, or even a combination of distinct capability requirements.
- Conversely, queries that require the similar underlying capabilities may be scattered across different semantic subsets due to superficial topical differences.



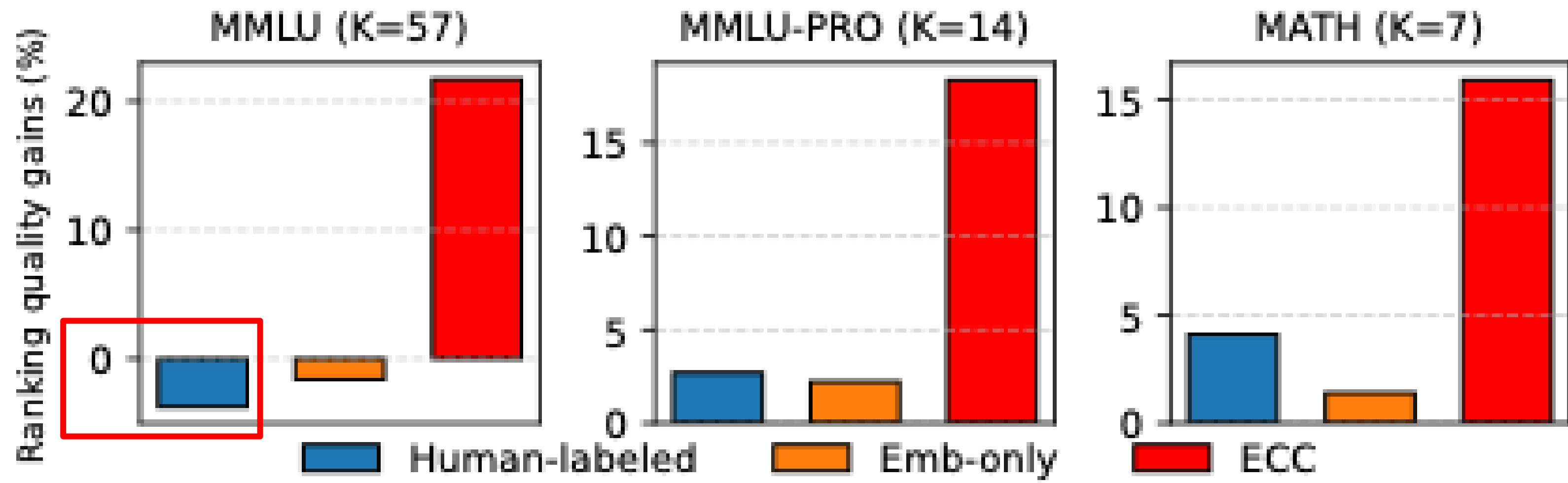
Semantics Often Fail to Reflect Underlying Capabilities!

- This (human-labeled) misalignment with the true capability distributions can limit, or even degrade, the generalizability of model capability estimates to unseen queries.



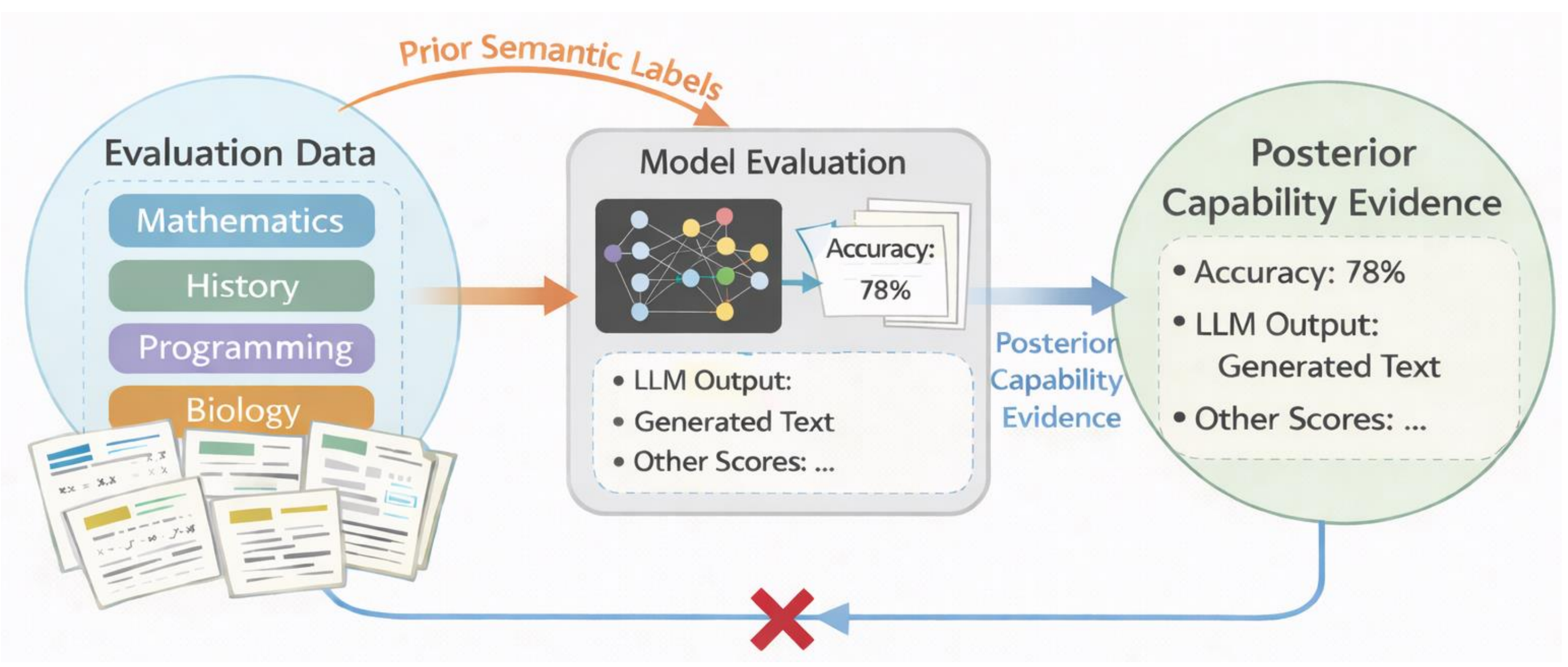
Semantics Often Fail to Reflect Underlying Capabilities!

- This (human-labeled) misalignment with the true capability distributions can limit, or even degrade, the generalizability of model capability estimates to unseen queries.
- **Goal**: Propose capability-aligned subsets (ECC) can better track the underlying capability structure, leading to more reliable and generalizable capability estimates on unseen queries.



From Semantics to Capability Evidence

- Key observation: Evaluation data is split and organized only by ***prior*** semantics, while ***posterior*** evidence of model capability is only used ***after model evaluation***.



Bradley-Terry Model — A Signal of Model Capability

- Widely used to estimate latent strength scores of items.
- Adopted to rank LLMs with pairwise model comparisons over query responses.
- Formally, let M be number of LLMs. The BT model parameterizes their strengths by a vector $\boldsymbol{\theta} \in \mathbb{R}^M$.
- The probability that model m_j is preferred over m_i is then defined using the logistic function $\sigma(\cdot)$:

$$P_{\boldsymbol{\theta}}(m_j \succ m_i) = \sigma(\theta_j - \theta_i) = \frac{1}{1 + e^{-(\theta_j - \theta_i)}}$$

Bradley-Terry Model — A Signal of Model Capability

- Let Q denote the set of evaluation queries.
- For each $q \in Q$, we observe a set of pairwise comparison records $\mathcal{N}_q = \{(i, j, y)\}$, where each tuple (i, j, y) encodes the outcome of comparing the responses of models m_i and m_j on query q.

Bradley-Terry Model — A Signal of Model Capability

- Let Q denote the set of evaluation queries.
- For each $q \in Q$, we observe a set of pairwise comparison records $\mathcal{N}_q = \{(i, j, y)\}$, where each tuple (i, j, y) encodes the outcome of comparing the responses of models m_i and m_j on query q.
- Define $y = 1$ indicates that m_j is preferred over m_i ,and $y = 0$ otherwise, then one can learn $\boldsymbol{\theta}$ by minimizing the BT model loss over pairwise comparisons across all queries:

$$\ell(\boldsymbol{\theta}) \ := \ -\sum_{q \in \mathcal{Q}} \sum_{(i,j,y) \in \mathcal{N}_q} \log P(y \mid i, j; \boldsymbol{\theta}),$$

where the log-probability $\log P(y \mid i, j; \ \boldsymbol{\theta})$ is defined as:

$$y \log \sigma(\theta_j - \theta_i) + (1 - y) \log \sigma(\theta_i - \theta_j).$$

Capability-Aware Ranking Quality

- We quantify **the quality of a learned model capability ranking $\boldsymbol{\theta}$** for a query q by how well it explains the observed pairwise comparison outcomes, operationalized via the following **per-query BT loss**:

$$\ell_{\text{comp}}(q;\boldsymbol{\theta}) = -\frac{1}{|\mathcal{N}_q|}\sum_{(i,j,y)\in\mathcal{N}_q}\log P(y \mid i,j;\boldsymbol{\theta}).$$

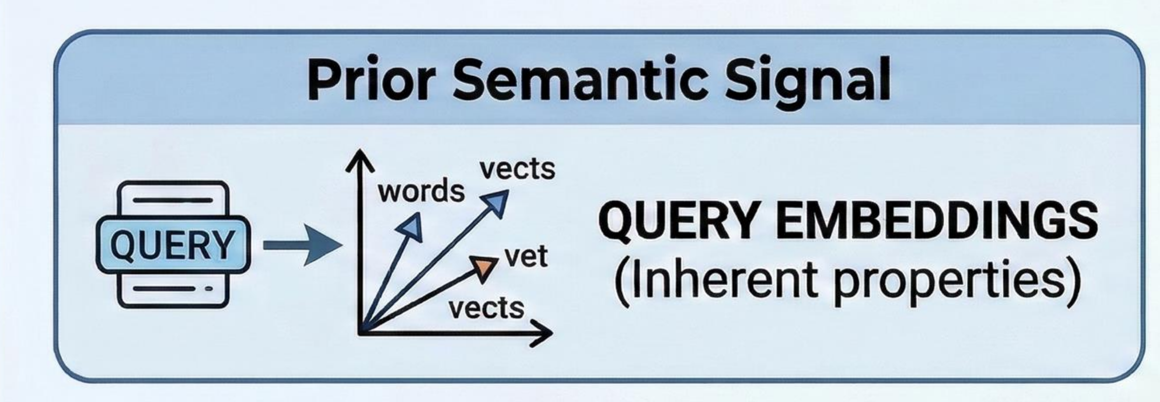
Capability-Aware Ranking Quality

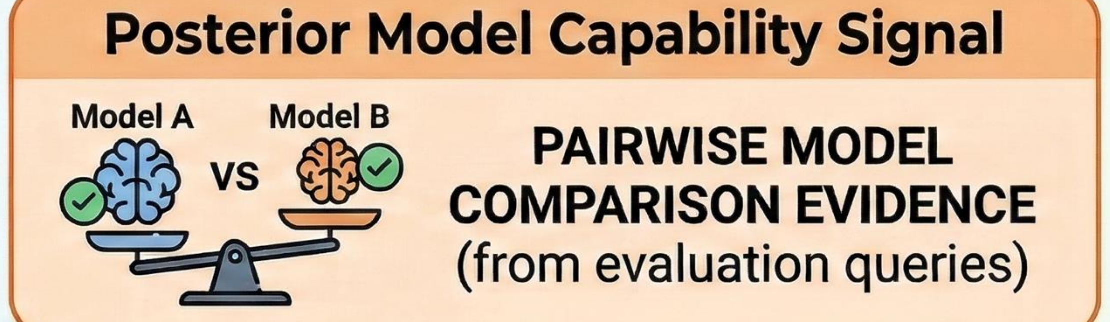
- We quantify **the quality of a learned model capability ranking $\boldsymbol{\theta}$** for a query q by how well it explains the observed pairwise comparison outcomes, operationalized via the following **per-query BT loss**:

$$\ell_{\text{comp}}(q; \boldsymbol{\theta}) = -\frac{1}{|\mathcal{N}_q|} \sum_{(i,j,y) \in \mathcal{N}_q} \log P(y \mid i, j; \boldsymbol{\theta}).$$

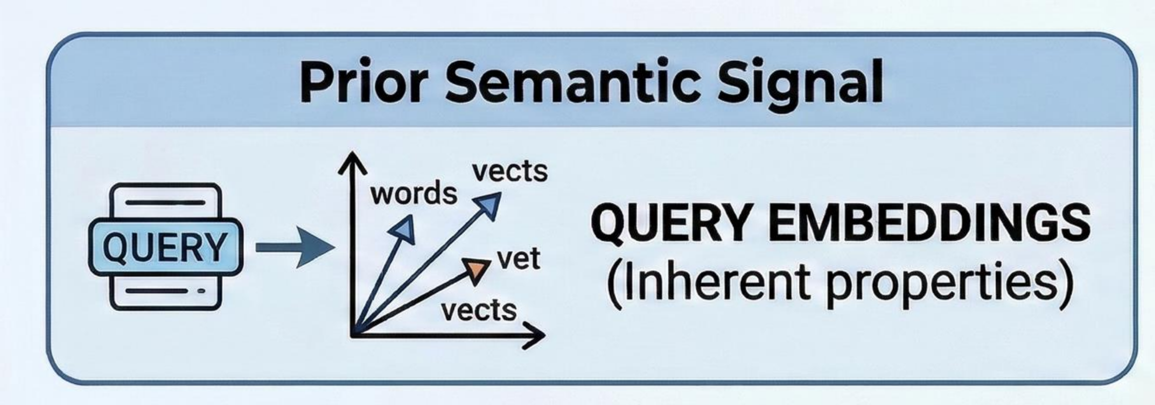
- Lower values of $\ell_{comp}(q; \boldsymbol{\theta})$ indicate a better fit to the observed evidence and thus higher ranking quality.
- This metric **is probability-aware**: it evaluates not only the ordinal consistency of the ranking with observed comparisons but also the predictive.

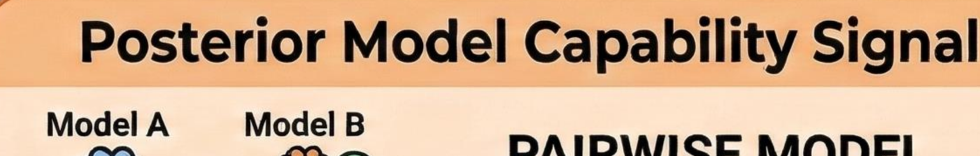
Two Types of Signals

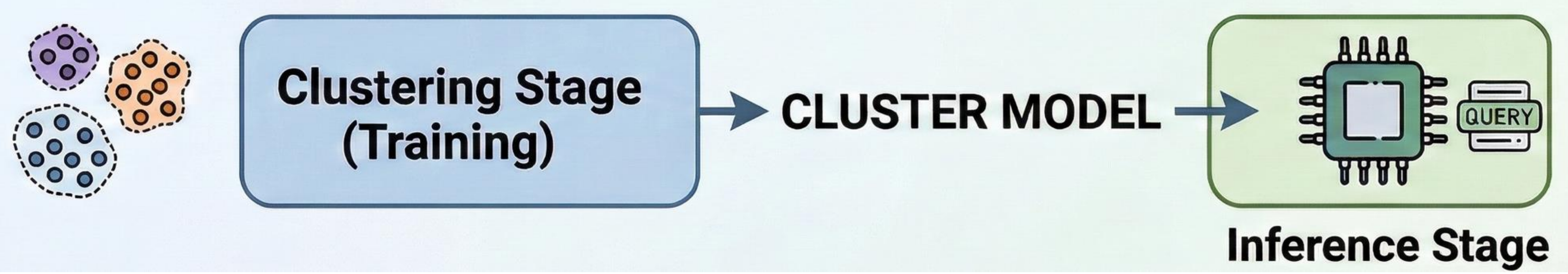




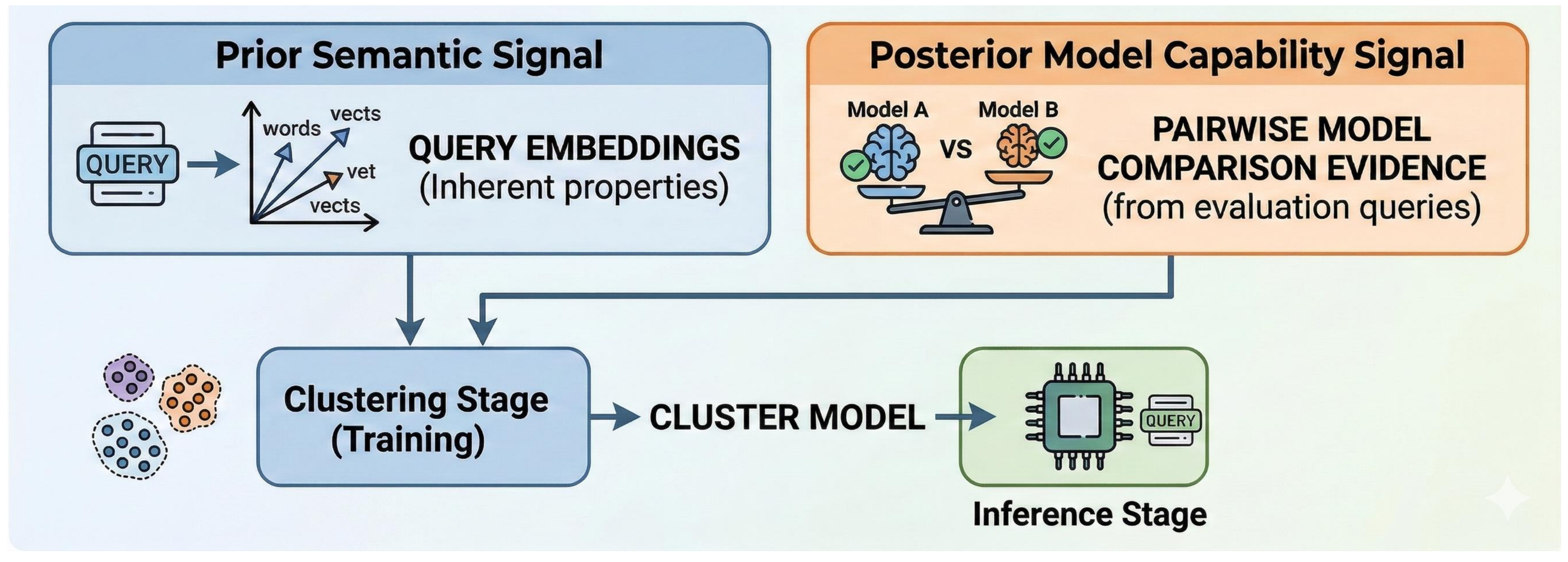
Two Stages



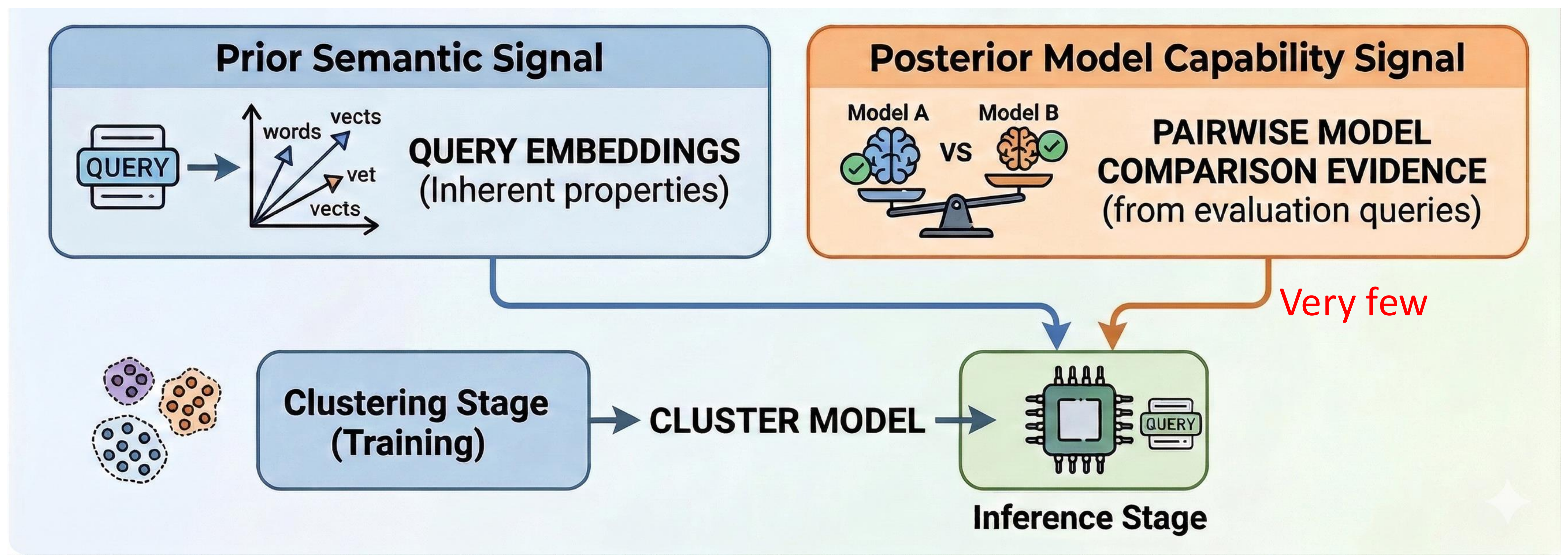




Evidence-Calibrated Clustering (ECC)



Evidence-Calibrated Clustering (ECC)



Evidence-Calibrated Clustering (ECC)

- To incorporate capability evidence during data clustering, ***ECC evaluates first, then clusters***.

Evidence-Calibrated Clustering (ECC)

- To incorporate capability evidence during data clustering, ***ECC evaluates first, then clusters***.
  - first collects pairwise comparison evidence (**posterior signal**) on unclustered queries,

Evidence-Calibrated Clustering (ECC)

- To incorporate capability evidence during data clustering, ***ECC evaluates first, then clusters***.
  - first collects pairwise comparison evidence (**posterior signal**) on unclustered queries,
  - then uses it to calibrate the embedding-based semantic clustering structure (**prior signal**) into capability-aware clusters.

Evidence-Calibrated Clustering (ECC)

- To incorporate capability evidence during data clustering, ***ECC evaluates first, then clusters***.
  - first collects pairwise comparison evidence (**posterior signal**) on unclustered queries,
  - then uses it to calibrate the embedding-based semantic clustering structure (**prior signal**) into capability-aware clusters.

- ECC characterizes each cluster $k \in [K]$ using **both two signals**:
  - **(i) Semantic centroid $\boldsymbol{c}_k$**: captures ***prior*** semantic structure in the embedding space.

Evidence-Calibrated Clustering (ECC)

- To incorporate capability evidence during data clustering, ***ECC evaluates first, then clusters***.
  - first collects pairwise comparison evidence (**posterior signal**) on unclustered queries,
  - then uses it to calibrate the embedding-based semantic clustering structure (**prior signal**) into capability-aware clusters.

- ECC characterizes each cluster $k \in [K]$ using **both two signals**:
  - **(i) Semantic centroid $\boldsymbol{c}_k$**: captures ***prior*** semantic structure in the embedding space.
  - **(ii) Capability profile $\boldsymbol{\theta}^{(k)}$** :BT model representing ***posterior*** strengths of the M LLMs on queries in cluster k.

Evidence-Calibrated Clustering (ECC)

- To incorporate capability evidence during data clustering, ***ECC evaluates first, then clusters***.
  - first collects pairwise comparison evidence (**posterior signal**) on unclustered queries,
  - then uses it to calibrate the embedding-based semantic clustering structure (**prior signal**) into capability-aware clusters.

- ECC characterizes each cluster $k \in [K]$ using **both two signals**:
  - **(i) Semantic centroid $\boldsymbol{c}_k$**: captures ***prior*** semantic structure in the embedding space.
  - **(ii) Capability profile $\boldsymbol{\theta}^{(k)}$** :BT model representing ***posterior*** strengths of the M LLMs on queries in cluster k.
  - Overall, define system parameters $\boldsymbol{\Theta} = \{(\boldsymbol{c}_k, \boldsymbol{\theta}^{(k)})\}_{k=1}^{K}$

Evidence-Calibrated Clustering (ECC)

- **Observation**: A query may encompass multiple capability requirements.

Evidence-Calibrated Clustering (ECC)

- **Observation**: A query may encompass multiple capability requirements.

- **Responsibilities** $\mathcal{P}$: ECC introduces soft trainable responsibility $\mathcal{P} = \{r_{qk}\}$ to capture capability mixtures across queries:
  - r_{qk} denotes the probability that q aligns with the capability of cluster k.
  - s.t. $r_{qk} \geq 0$, and $\sum_{k=1}^{K} r_{qk} = 1$.

Evidence-Calibrated Clustering (ECC)

- **Alternating Optimization:** ECC iteratively updates (i) the system parameters $\boldsymbol{\Theta}$ and (ii) the soft responsibilities $\mathcal{P}$.

- **Update $\boldsymbol{\Theta}$ (given $\boldsymbol{\mathcal{P}}$):**
  - **Centroids $\boldsymbol{c}_k$**: for each cluster k, update the embedding centroid c_k as the responsibility-weighted average of all query embeddings.
  - **Capability profile $\boldsymbol{\theta}^{(k)}$**: for each cluster k, train the BT model $\boldsymbol{\theta}^{(k)}$ by minimizing the responsibility-weighted BT loss $\ell_{comp}\left(q; \boldsymbol{\theta}^{(k)}\right)$ over all queries.

Evidence-Calibrated Clustering (ECC)

- **Alternating Optimization:** ECC iteratively updates (i) the system parameters $\boldsymbol{\Theta}$ and (ii) the soft responsibilities $\mathcal{P}$.

- **Update $\boldsymbol{\Theta}$ (given $\mathcal{P}$):**
  - **Centroids $\boldsymbol{c}_k$**: for each cluster k, update the embedding centroid c_k as the responsibility-weighted average of all query embeddings.
  - **Capability profile $\boldsymbol{\theta}^{(k)}$**: for each cluster k, train the BT model $\boldsymbol{\theta}^{(k)}$ by minimizing the responsibility-weighted BT loss $\ell_{comp}(q; \boldsymbol{\theta}^{(k)})$ over all queries.
- **Update $\mathcal{P}$ (given $\boldsymbol{\Theta}$):**
  - For each query q, update its responsibility r_{qk} by combining BT-model agreement (BT loss) and embedding closeness (similarity) to cluster k.

Evidence-Calibrated Clustering (ECC)

- **Alternating Optimization:** ECC iteratively updates (i) the system parameters $\boldsymbol{\Theta}$ and (ii) the soft responsibilities $\mathcal{P}$.

- **Update Θ (given $\mathcal{P}$):**
  - **Centroids $\boldsymbol{c}_k$**: for each cluster k, update the embedding centroid c_k as the responsibility-weighted average of all query embeddings.
  - **Capability profile $\boldsymbol{\theta}^{(k)}$**: for each cluster k, train the BT model $\boldsymbol{\theta}^{(k)}$ by minimizing the responsibility-weighted BT loss $\ell_{comp}(q; \boldsymbol{\theta}^{(k)})$ over all queries.
- **Update $\mathcal{P}$ (given Θ):**
  - For each query q, update its responsibility r_{qk} by combining BT-model agreement (BT loss) and embedding closeness (similarity) to cluster k.
- **Convergence**: when $\mathcal{P}$ and $\boldsymbol{c}_k$ stabilize.

Probe-Informed Mixture Inference

- Inferring LLM capabilities for a new query q requires its soft weights $\boldsymbol{r}_q$ across clusters.
- Using embedding along is not optimal:
  - Semantic proximity $\nRightarrow$ capability proximity
  - Embedding-only inference mismatches training signals (embedding + evidence)

Probe-Informed Mixture Inference

- Inferring LLM capabilities for a new query q requires its soft weights $\boldsymbol{r}_q$ across clusters.
- Using embedding along is not optimal:
  - Semantic proximity $\nRightarrow$ capability proximity
  - Embedding-only inference mismatches training signals (embedding + evidence)
- Lightweight inference with single probe:
  - At inference, collect **one random pairwise model comparison as a probe**, and combine it with the embedding to estimate $\boldsymbol{r}_q$.

Probe-Informed Mixture Inference

- Inferring LLM capabilities for a new query q requires its soft weights $\boldsymbol{r}_q$ across clusters.
- Using embedding along is not optimal:
  - Semantic proximity $\nRightarrow$ capability proximity
  - Embedding-only inference mismatches training signals (embedding + evidence)
- Lightweight inference with single probe:
  - At inference, collect **one random pairwise model comparison as a probe**, and combine it with the embedding to estimate $\boldsymbol{r}_q$.
  - With $\boldsymbol{r}_q$, ECC estimates pairwise preference by a weighted mixture:

$$P(m_j \succ m_i \mid q) = \sum_{k=1}^{K} r_{qk} \sigma \left(\boldsymbol{\theta}_j^{(k)} - \boldsymbol{\theta}_i^{(k)} \right).$$

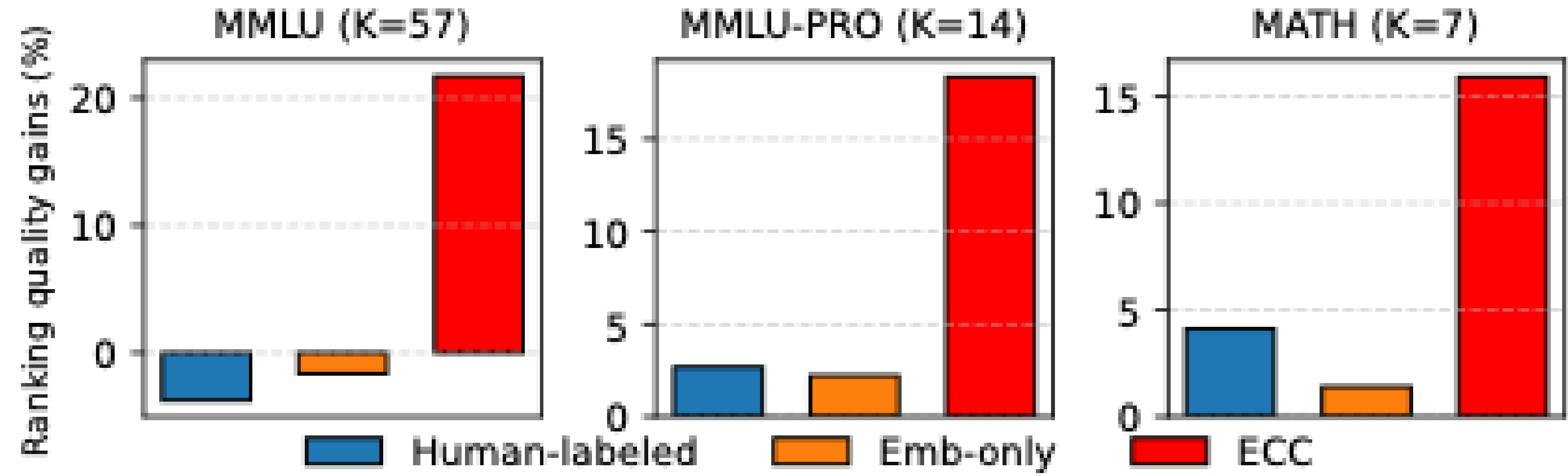
Quantitative Results

- **Metric**: Ranking quality gains.
  - defined as the **average per-query reduction in BT loss** on pair-wise model comparisons for unseen queries, compared to a single global BT model trained without clustering.

Quantitative Results

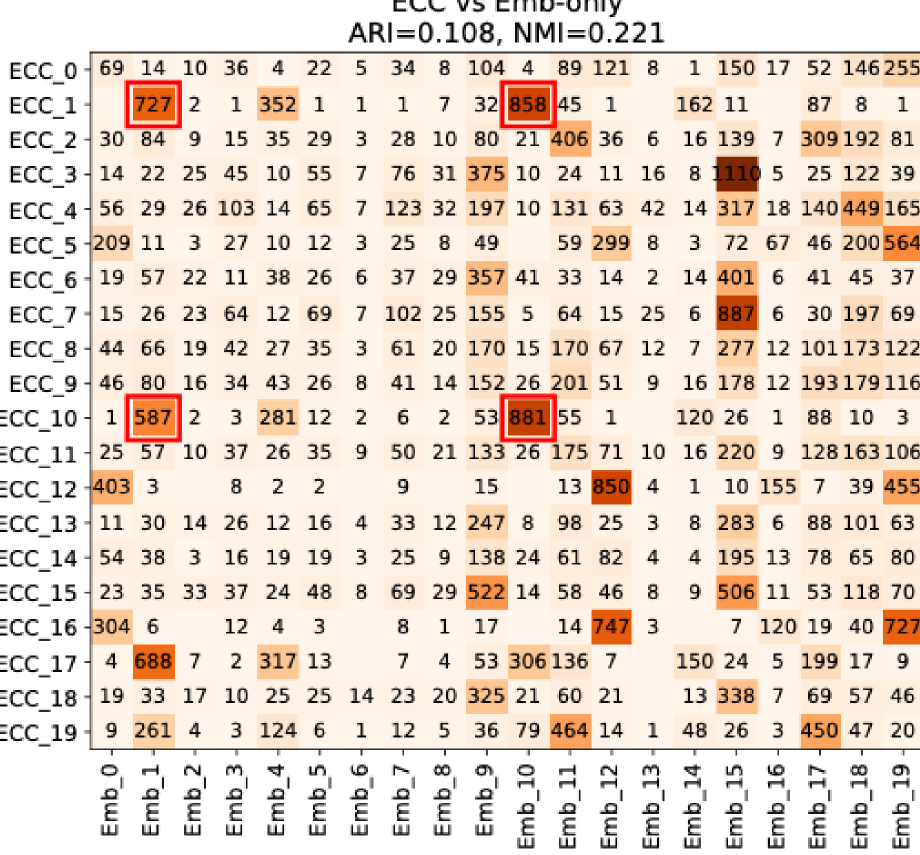
- **Metric**: Ranking quality gains.
  - defined as the **average per-query reduction in BT loss** on pair-wise model comparisons for unseen queries, compared to a single global BT model trained without clustering.

- **RQ**: How does ECC compare to human-labeled data split/clustering?



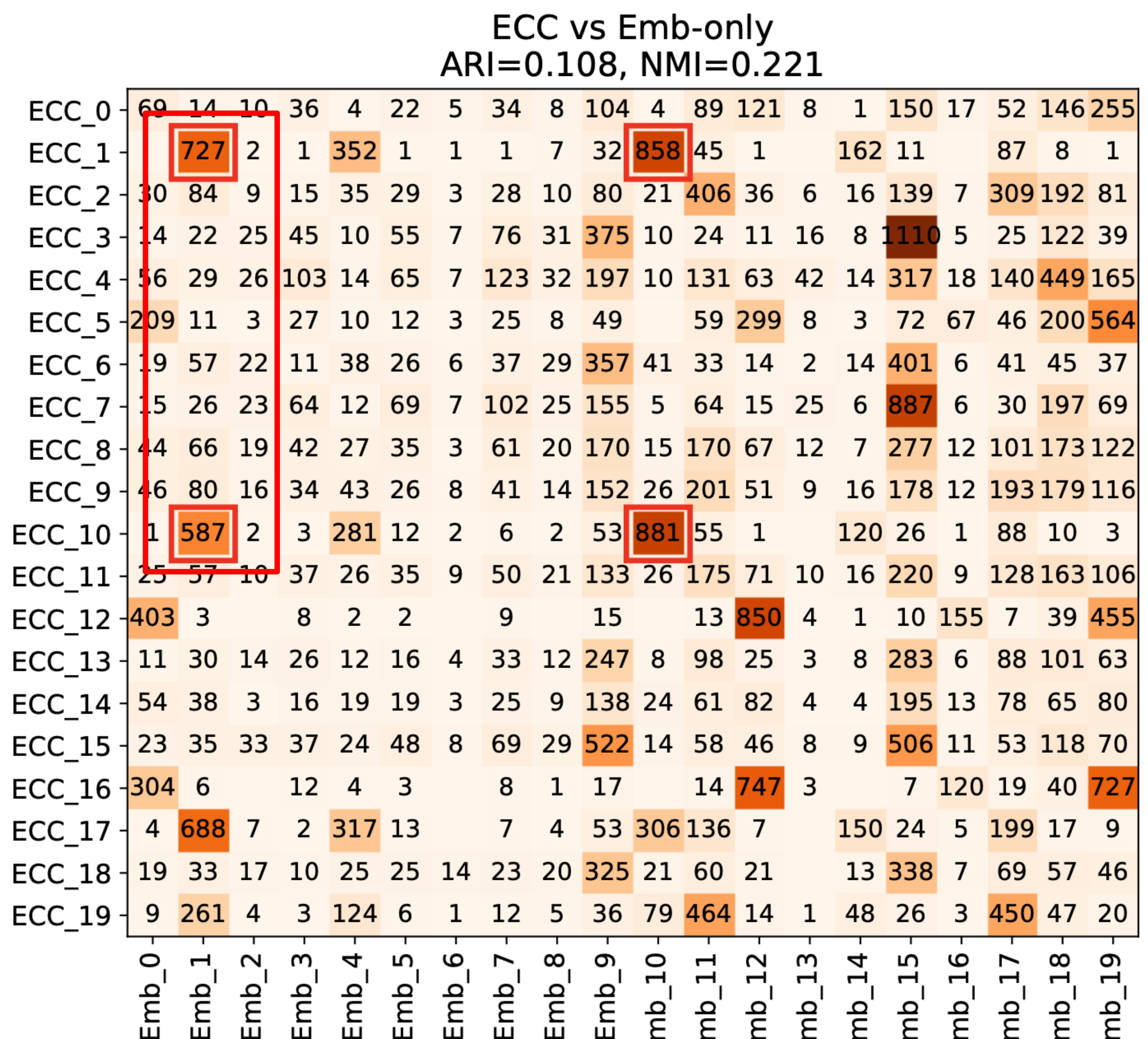
Cluster Overlap Qualitative Analysis

- We present the **pairwise cluster overlap matrices of hard partitions** induced by two clustering methods when the number of clusters is 20.



Within-Embedding Split

- **Semantic**:
  - Both cells fall into an advanced chemistry/biochemistry region.
- **Capability**:
  - *ECC_1 ∩ Emb_1* emphasizes evidence-grounded inference from analytical or computational signals.
  - *ECC_10 ∩ Emb_1* focuses on constraint-aware design and procedure planning on top of mechanistic reasoning with explicit safety/specificity constraints



Within-Embedding Split

- ECC_1 ∩ Emb_1
    - *“Identify the unknown organic compound (molecular formula C10H12O) using infrared spectroscopy and proton NMR spectroscopy data interpretation.”*
    - *“How does the surface energy of a metal change when a foreign atom is adsorbed onto the surface, based on Density Functional Theory calculations?”*

- ECC_10 ∩ Emb_1
    - *“How can we design a drug that specifically targets the cell wall of gram-negative bacteria, such as Escherichia coli, without affecting the cells of the human host?”*
    - *“How can you design a drug that specifically targets the enzyme Cyclooxygenase-2 (COX-2) without affecting the related enzyme Cyclooxygenase-1 (COX-1)?”*

Cross-Embedding Merge

- **Semantic**:
  - *ECC_1 ∩ Emb_1* focuses on biochemistry.
  - *ECC_1 ∩ Emb_10* focuses on materials property.

- **Capability**:
  - both cells have the same capability demand: parameter-to-outcome causal modeling with simulation-supported prediction.

ECC vs Emb-only
ARI=0.108, NMI=0.221

	Emb_0	Emb_1	Emb_2	Emb_3	Emb_4	Emb_5	Emb_6	Emb_7	Emb_8	Emb_9	Emb_10	Emb_11	Emb_12	Emb_13	Emb_14	Emb_15	Emb_16	Emb_17	Emb_18	Emb_19
ECC_0	69	14	10	36	4	22	5	34	8	104	4	89	121	8	1	150	17	52	146	255
ECC_1		727	2	1	352	1	1	1	7	32	858	45	1		162	11		87	8	1
ECC_2	30	84	9	15	35	29	3	28	10	80	21	406	36	6	16	139	7	309	192	81
ECC_3	14	22	25	45	10	55	7	76	31	375	10	24	11	16	8	1110	5	25	122	39
ECC_4	56	29	26	103	14	65	7	123	32	197	10	131	63	42	14	317	18	140	449	165
ECC_5	209	11	3	27	10	12	3	25	8	49		59	299	8	3	72	67	46	200	564
ECC_6	19	57	22	11	38	26	6	37	29	357	41	33	14	2	14	401	6	41	45	37
ECC_7	15	26	23	64	12	69	7	102	25	155	5	64	15	25	6	887	6	30	197	69
ECC_8	44	66	19	42	27	35	3	61	20	170	15	170	67	12	7	277	12	101	173	122
ECC_9	46	80	16	34	43	26	8	41	14	152	26	201	51	9	16	178	12	193	179	116
ECC_10	1	587	2	3	281	12	2	6	2	53	881	55	1		120	26	1	88	10	3
ECC_11	25	57	10	37	26	35	9	50	21	133	26	175	71	10	16	220	9	128	163	106
ECC_12	403	3		8	2	2		9		15		13	850	4	1	10	155	7	39	455
ECC_13	11	30	14	26	12	16	4	33	12	247	8	98	25	3	8	283	6	88	101	63
ECC_14	54	38	3	16	19	19	3	25	9	138	24	61	82	4	4	195	13	78	65	80
ECC_15	23	35	33	37	24	48	8	69	29	522	14	58	46	8	9	506	11	53	118	70
ECC_16	304	6		12	4	3		8	1	17		14	747	3		7	120	19	40	727
ECC_17	4	688	7	2	317	13		7	4	53	306	136	7		150	24	5	199	17	9
ECC_18	19	33	17	10	25	25	14	23	20	325	21	60	21		13	338	7	69	57	46
ECC_19	9	261	4	3	124	6	1	12	5	36	79	464	14	1	48	26	3	450	47	20

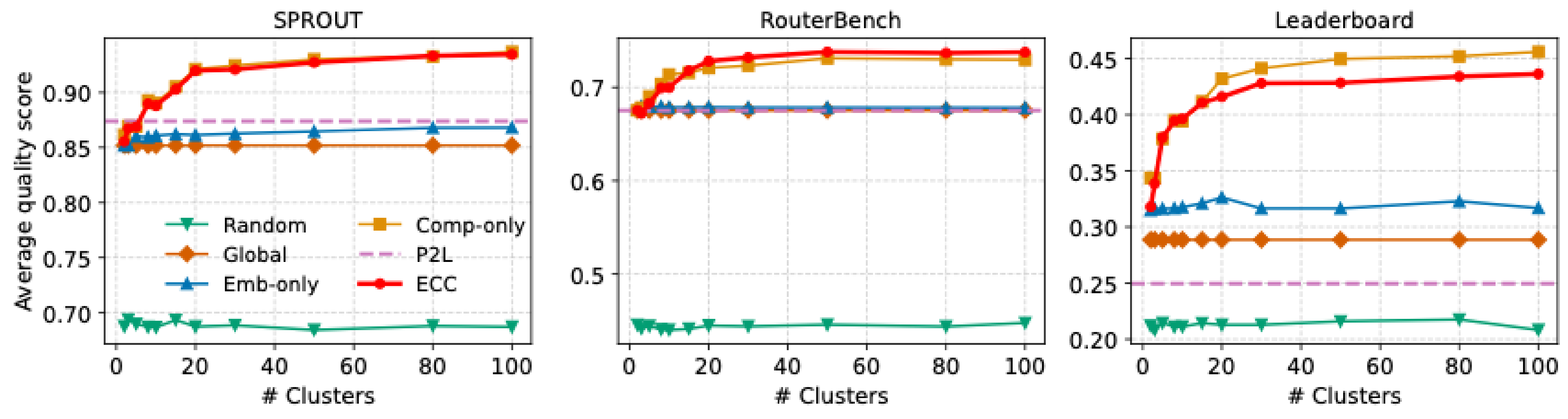
Cross-Embedding Merge

- *ECC_1 ∩ Emb_1*
  - *“How does the surface energy of a metal change when a foreign atom is adsorbed onto the surface, based on Density Functional Theory calculations?”*
  - *“How does temperature affect the phase transition of water from solid (ice) to liquid using Monte Carlo simulations in computational chemistry?”*

- *ECC_1 ∩ Emb_10*
  - *“What are the effects of doping on the electrical and optical properties of semiconductors, and how does the dopant concentration affect these properties?”*
  - *“How does the electronic transport property (conductivity) of a graphene nanoribbon change as the width of the ribbon is varied, as predicted by density functional theory (DFT) calculations?”*

Application: Query Routing

- Goal: route each query to the model that yields the highest response quality.



Thanks! Question?