NEURAL INFORMATION PROCESSING SYSTEMS

Transcending Cost-Quality Tradeoff in Agent Serving Systems via Session-Awareness

Yanyu Ren[1], Li Chen[2], Dan Li[1,2], Xizheng Wang[1], Zhiyuan Wu[1], Yukai Miao[2], Yu Bai[2]

[1]Tsinghua University, [2]Zhongguancun Laboratory

Agentic workflow is a powerful tool for solving real-world problems.

- The agentic workflow can continuously optimize and improve the model output through the agent's observation of the "environment" in each round of interaction, and finally complete a difficult task.
- The powerful general model and the finely tuned domain-specific model have strong capabilities and can solve a variety of complex problems such as household robots, intelligent operating systems, and intelligent information retrieval.

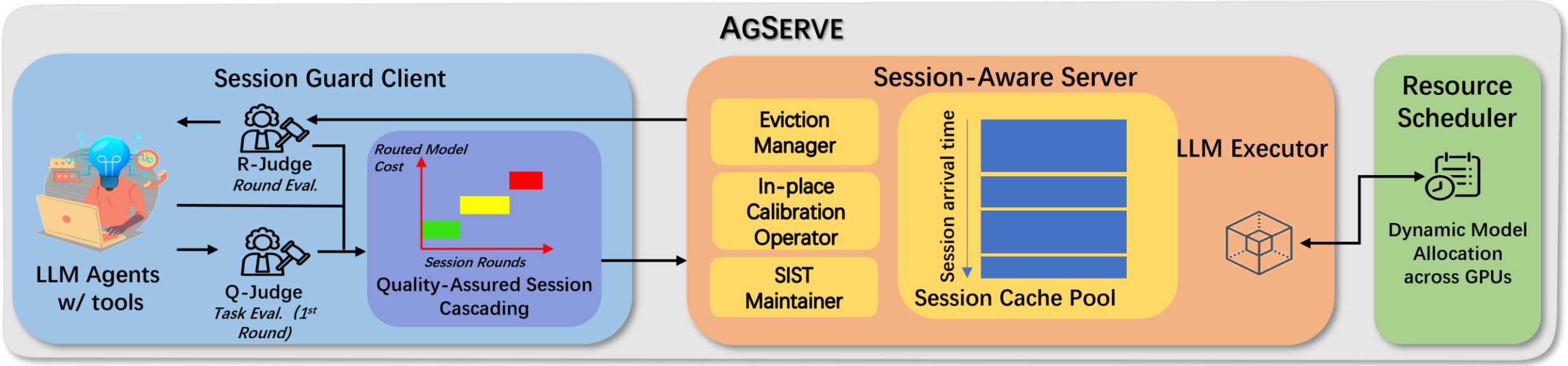
Characteristics of agentic workflow

- Agents use automated tools to obtain observations from the environment **within a few to a few hundred milliseconds** and request the LLM to further revise the answer and provide actions. Humans, on the other hand, typically need to think for tens of seconds or even several minutes before making the next request.
- In terms of inference time, the agent only needs to generate very short responses (**less than 100 tokens**), while humans would generate responses with thousands of tokens, putting more emphasis on prefilling.
- The prompt length grows as the session progresses, and may encounter “middle truncation”, which abandon the rounds of conversation following the system prompt.

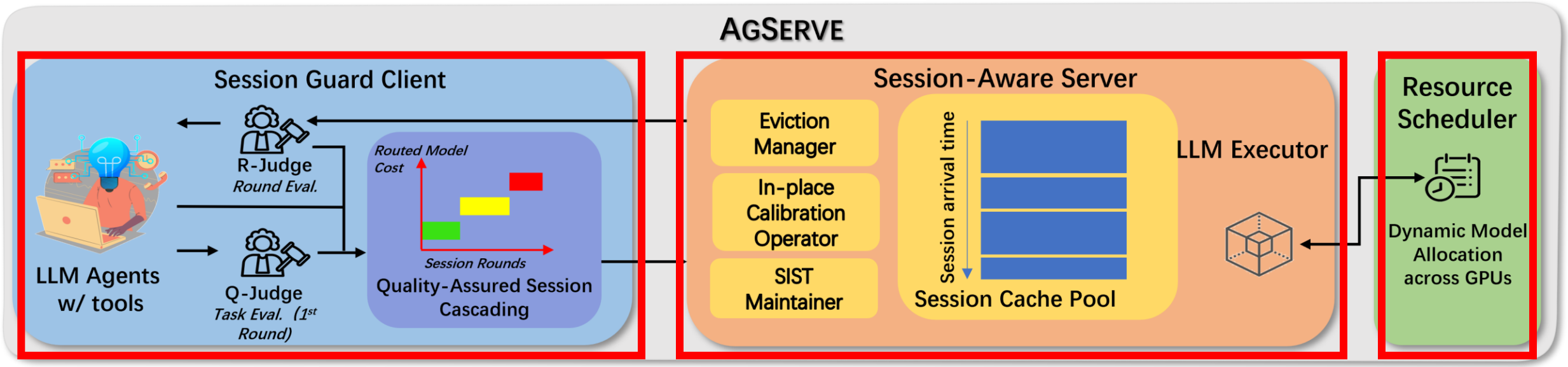
Transcending the Cost-Quality Tradeoff

- Cost-quality tradeoff is a limitation faced by traditional LLM serving systems that only provide one model service in the context of LLM agents.
  - If a small model is chosen, it will be difficult to solve complex problems after multiple iterations, resulting in a decline in service quality.
  - If a large model is chosen, it will result in significant overhead and increased costs in the initial rounds.
- Two key technologies to overcome the tradeoff
  - Focus on the *session* characteristic of agents. Use sessions to maintain cached data such as KV caches , reducing redundant computations and improving throughput.
  - Use cascading models of different sizes. Deploy models of different sizes on the cluster, allowing smaller models to solve simpler problems in the early stages and larger models to solve more complex problems in the later stages.

Design of AgServe



Design of AgServe



Minimize model/API cost (§ 5)

Maximize cache hit rate (§ 4)

Maximize hardware utilization (§ 6)

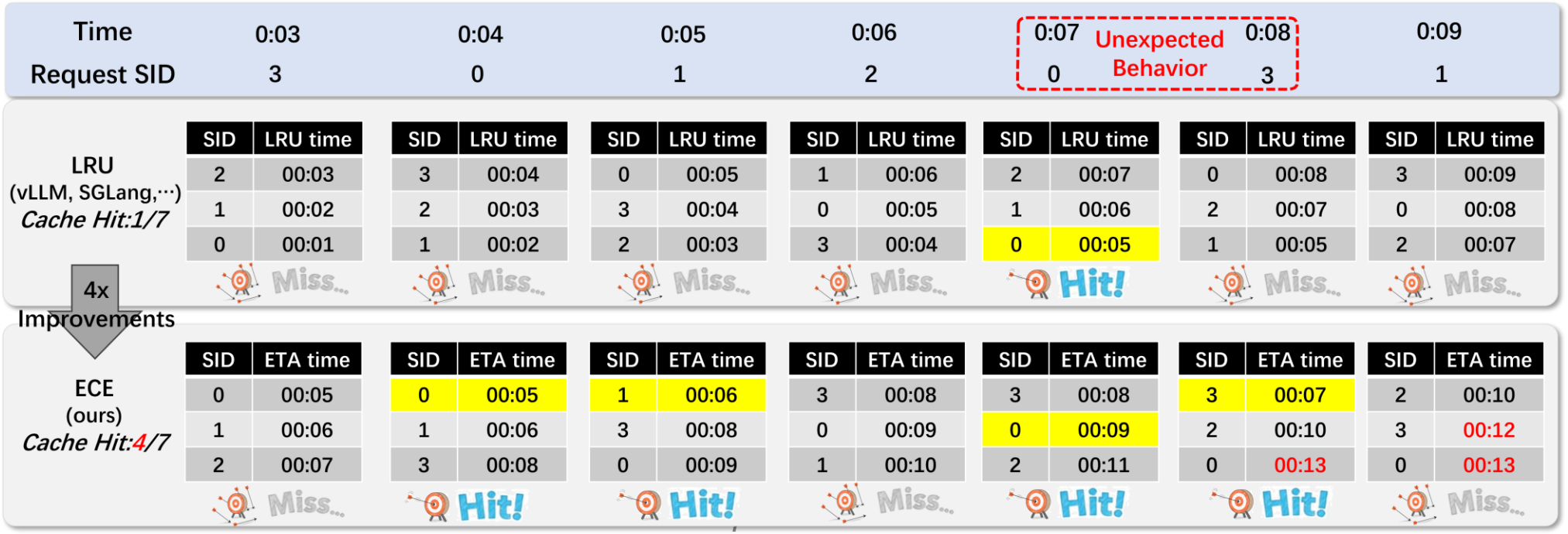
SAS calibrates KV cache for middle truncation

- **Background: Most serving systems use prefix matching to find the cache of previous request**
- When middle truncation happens, traditional methods that requires prefix matching will not work. This leads to low hit rate.
- SAS first tracks the session ID to find the KV cache, and abandons the KV caches of truncated tokens.
- As the positional embedding of the tokens are memorized inside the KV Cache, SAS calibrates the positional embedding with the following formula:

$$K'_{p',2i} = cos(-\delta_p\theta_i)K_{p,2i} + sin(-\delta_p\theta_i)K_{p,2i+1}$$
$$K'_{p',2i+1} = cos(-\delta_p\theta_i)K_{p,2i+1} - sin(-\delta_p\theta_i)K_{p,2i}$$

SAS evicts KV cache with Estimated-Time-of-Arrival

- **Background: Most serving systems use Least-Recent-Used algorithm to decide which session cache to evict.**
- The regularity of the request pattern of agent gives us an opportunity to estimate the arrival time of the next request in the session.
- SAS estimates the ETA with the history data, and chooses to evict the session that are estimated to arrive at the latest time.



Q-Judge decides the initial task difficulty in SGC

- In the first round of interaction in the session, a pre-trained task difficulty judge (Q-Judge) guides the session to a model of appropriate size in advance.
- The judge's deviation causes overkill (picking a larger model) and underkill (picking a smaller model), which introduces extra cost.
- Our observation shows that overkill is more harmful than underkill, since it is unrecoverable with an upgoing cascade.

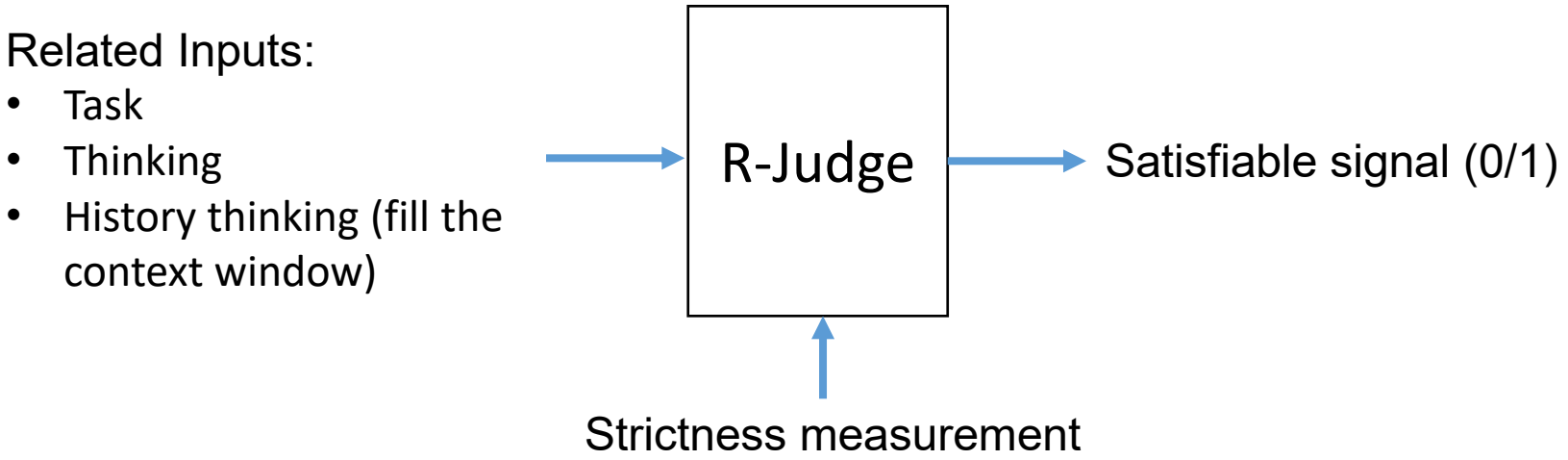
$$f(p,q,g) = -\sum_{i=1}^{n} p(x_i) log(q(x_i)) + \beta \cdot \sum_{i=1}^{n} (x_i - g)^2 \cdot \frac{\alpha_i}{1 - q(x_i)}$$

Cross-entropy loss

Penalty for deviation (α_i introduces more penalty for overkill)

R-Judge guides the serving quality in SGC

- In each subsequent round of interaction, SGC adopts one of the retry or migrate strategy to mitigate quality issues.
- Four typical types of quality issues in agent serving: service failure, violation of preset rules, invalid actions, and low reasoning quality.
- We pre-trained response quality judge (R-Judge) to evaluate the reasonableness of the model response, its relevance to the task, and other rules.



RS decides the allocation of devices onto each model

1. Dynamic Allocation Strategy

- RS tracks inference frequency to estimate model demand.
- Supply = available KV-cache tokens across instances.
- Models with high demand/supply ratio are scaled up; low-ratio ones scaled down.

2. Resource Redistribution

- Redistribute instances across nodes dynamically.
- Consolidate same-model instances for intra-model parallelism.
- Example: 7 GPUs with $w_m = 1$ → instances on 1, 2, and 4 GPUs.

Result: Higher GPU utilization • Reduced communication overhead • Adaptive scaling

Evaluations

- AgServe reduced service costs by 83.5% while maintaining similar quality to GPT-4o .
- At the same service cost, AgServe offers 1.8x the service quality .
- Under the load of multiple agent services, AgServe reduced service costs by 64% compared to Llumnix, while achieving 1.6x the quality.
- The ECE policy achieves up to 2.86x cache hit rate, and reduces the per-round duration by up to 50%.
- The dynamic allocation policy reduces 14% of P90 latency.

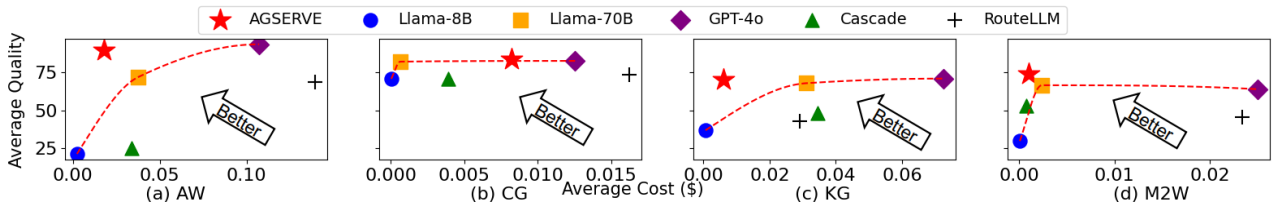


Figure: AgServe transcends the tradeoff curve

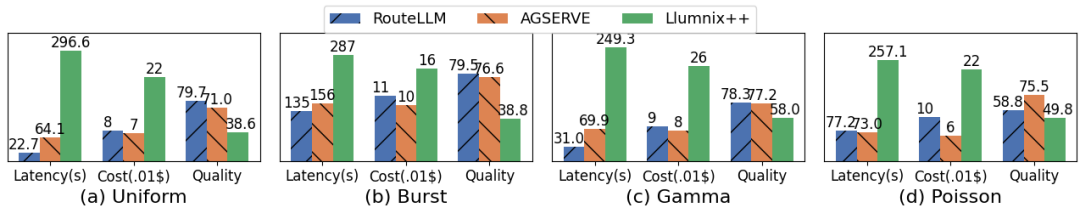


Figure: AgServe outperforms in multi-agent scenarios