DRAGON: Enhancing On-Device Model Performance with Distributed Retrieval-Augmented Generation

Shangyu Liu, Zhenzhe Zheng, Xiaoyao Huang†, Fan Wu, Guihai Chen, Jie Wu†*

Shanghai Jiao Tong University, † China Telecom Cloud Computing Research Institute, *Temple University

Oct. 27th, 2025

ACM MobiHoc 2025, Houston, USA

Content

Background & Motivation

Why We Need On-Device Language Models?
Why We Need Distributed Knowledge Database?
Distributed Retrieval and the Dilemma

Design

A Rough Solution
Problem Formulation
Speculative Aggregation
Greedy Scheduling
Theoretical Analysis
System Design

Experiments

Implementation
Evaluation

Conlusion

Background & Motivation

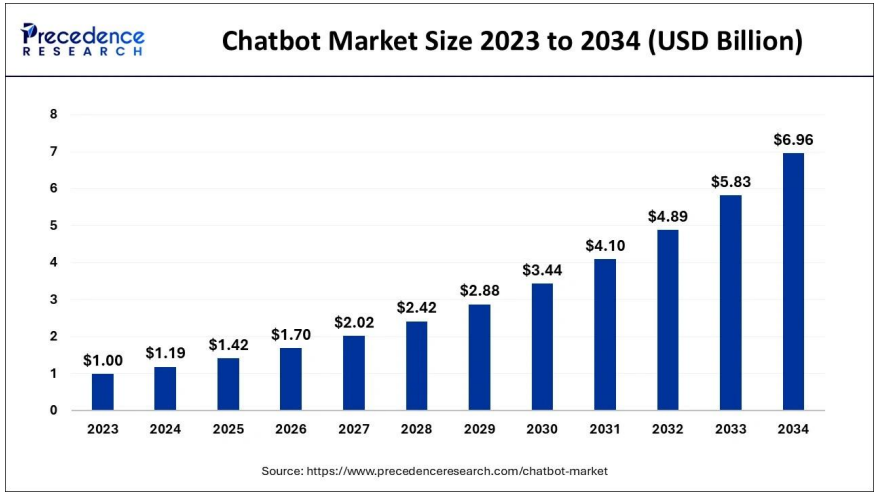
Background: Why We Need On-Device Language Models?

DeepSeek V3.2 | GPT 5

Grok 4 Heavy | Gemini 2.5 Pro

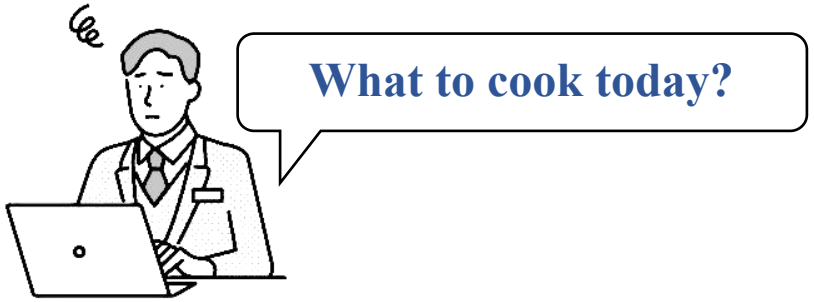
Claude Sonnet 4.5 | Kimi K2 Instruct

Large language models



chatbots growing in popularity rapidly

In LLM-based chatbot applications, some queries require integrating both **personalized** user data and **general** knowledge.



personalized data

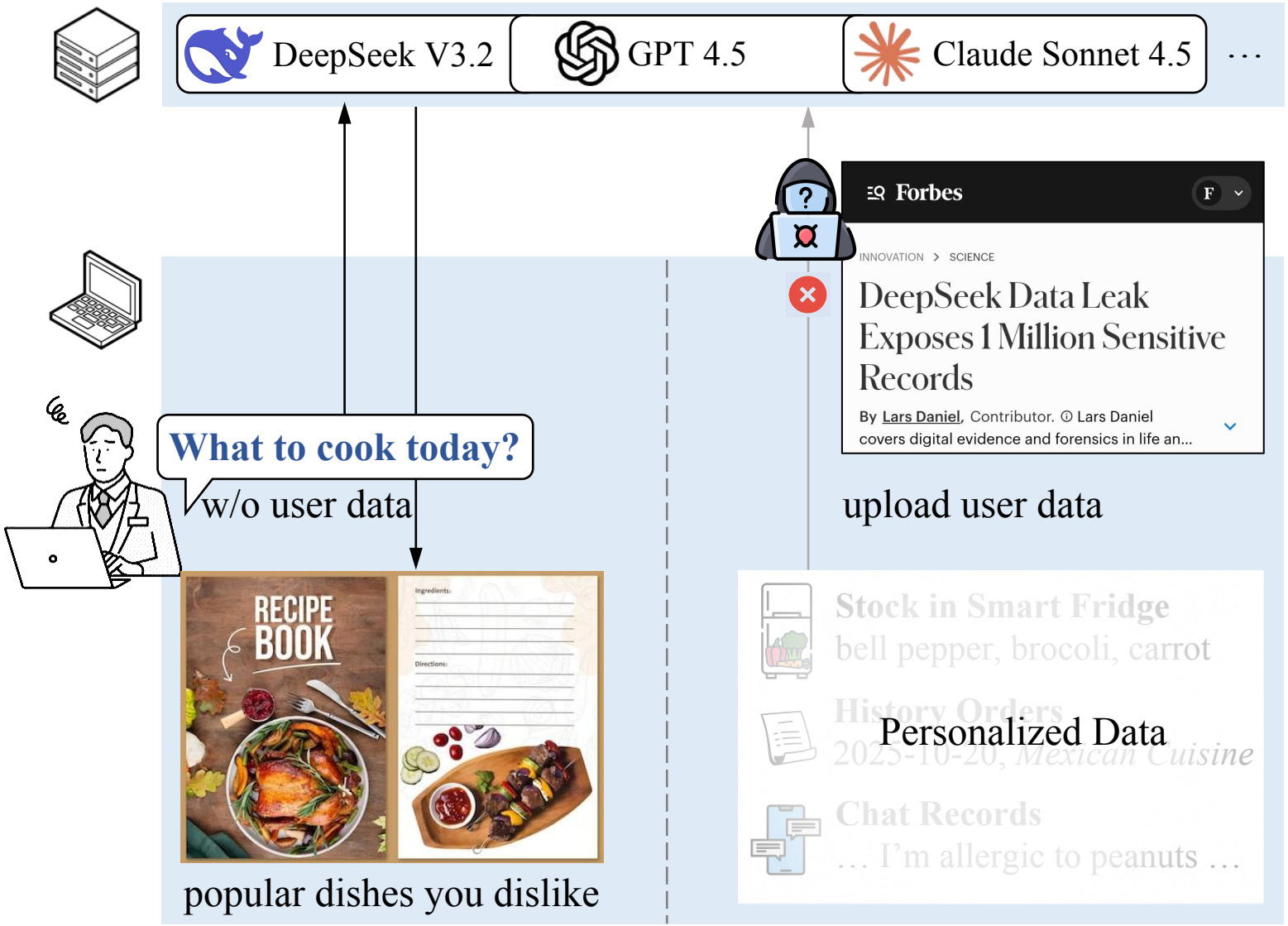
Stock in Smart Fridge
bell pepper, brocoli, carrot

History Orders
2025-10-20, *Mexican Cuisine*

Chat Records
… I'm allergic to peanuts …

general knowledge

Background: Why We Need On-Device Language Models?



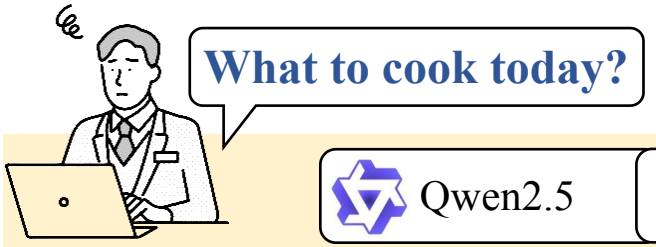
Cloud-hosted Large Language Models (LLMs)

w/o user data
– **lack of personalization**

upload user data
– **raising privacy concerns**

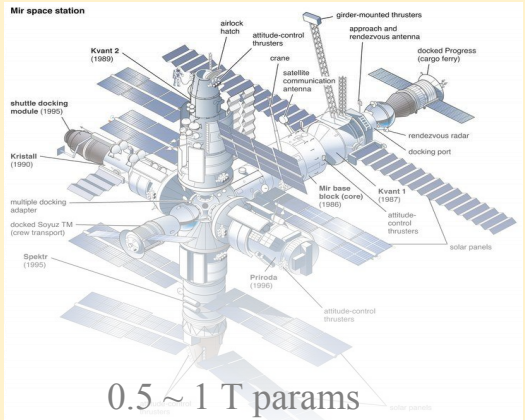
On-device Small Language Models (SLMs)

Background: Why We Need Distributed Knowledge Database?



Qwen2.5 | Phi-4 mini | Llama 3.2 | ⋯

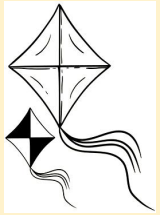
hardly support on-device supervised fine-tuning (SFT)



0.5 ~ 1 T params

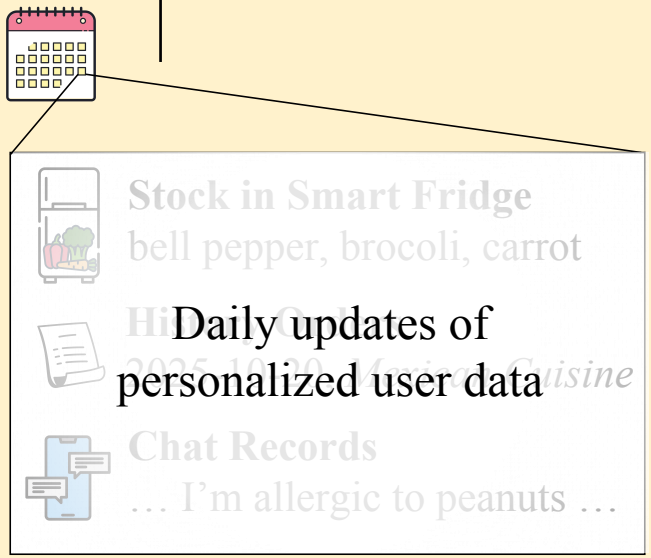
LLM is like a space station

Performance gap



1 ~ 4 B params

SLMs is like a kite



Daily updates of personalized user data

On-Device Small Language Models (SLMs)

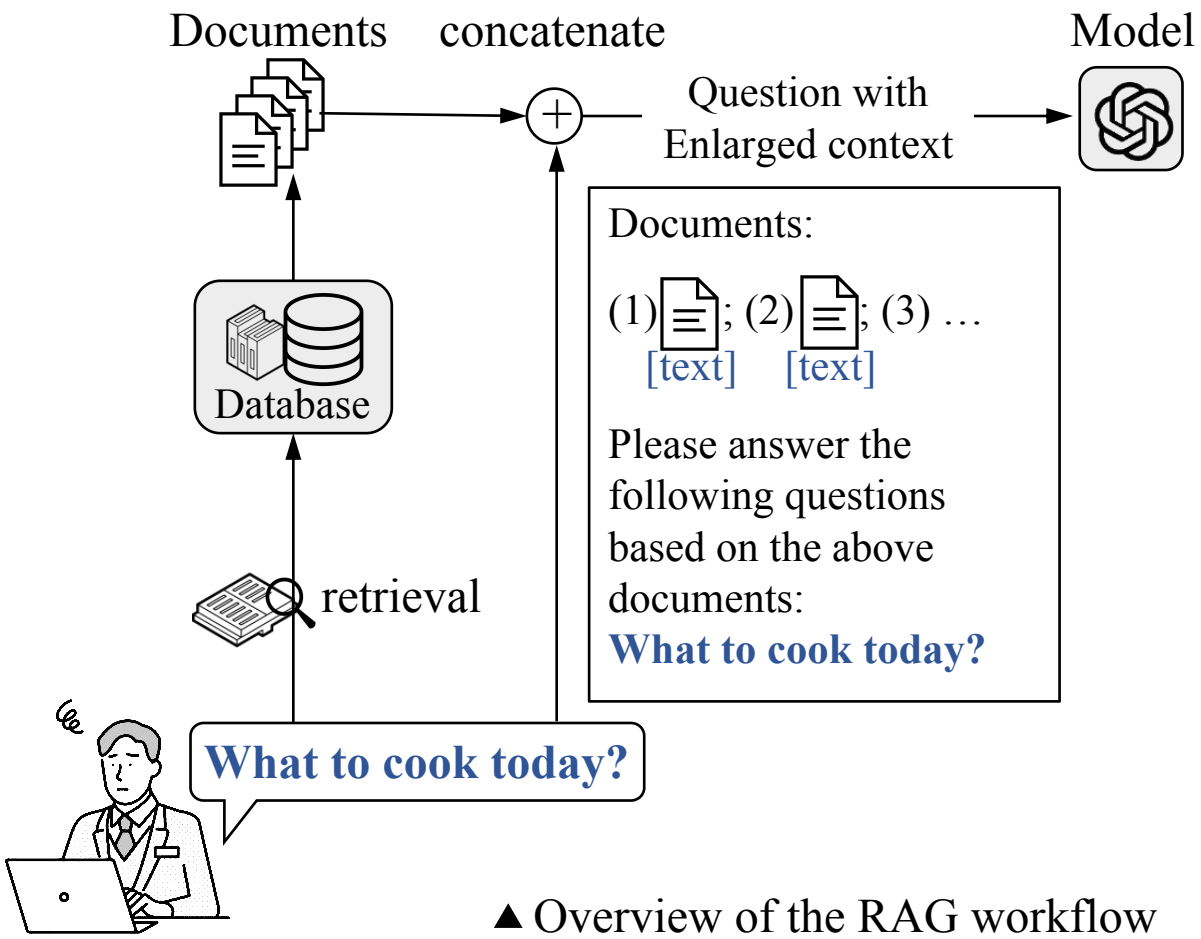
Data evolves continuously
– **cannot afford training**

Reduced model parameters
– **performance degradation**

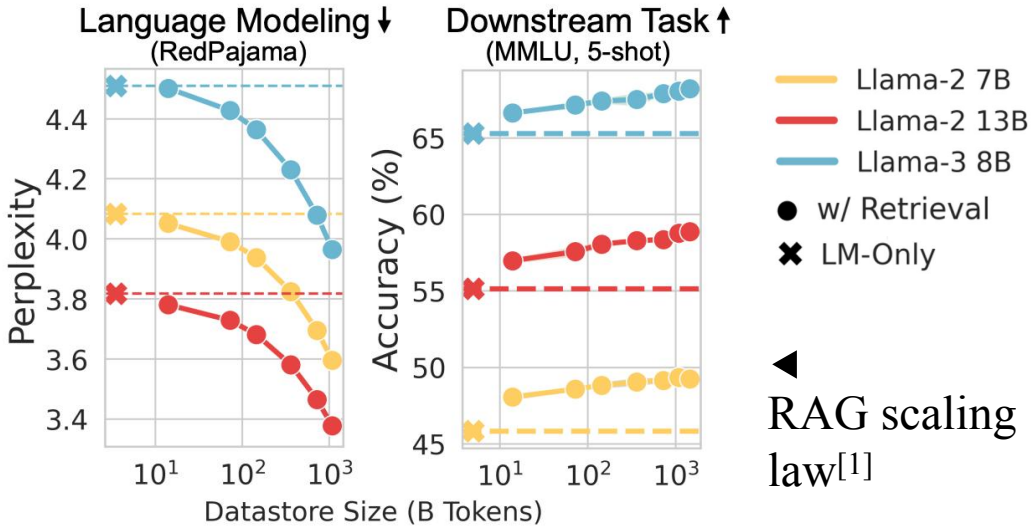
Retrieval-Augmented Generation (RAG)

Background: Why We Need Distributed Knowledge Database?

Retrieval-Augmented Generation (RAG)



▲ Overview of the RAG workflow



◀ RAG scaling law[1]

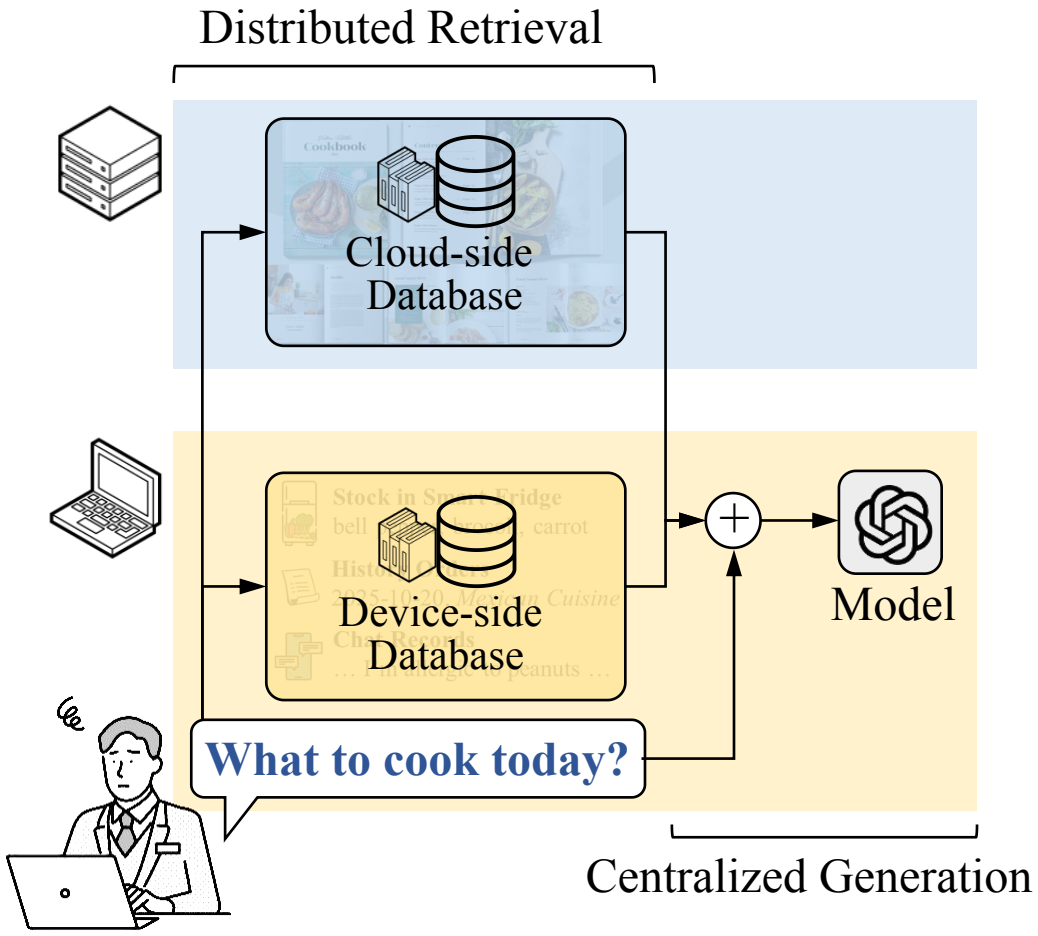
(1) generation **quality improves consistently** with the scale of the retrieval database[1] (require a large database to enhance performance)

(2) the database supports **convenient updates** (require a private database that updates frequently)

[1] Shao, Rulin, et al. "Scaling retrieval-based language models with a trillion-token datastore." *Advances in Neural Information Processing Systems* 37 (2024): 91260-91299.

Background: Distributed Retrieval and the Dilemma

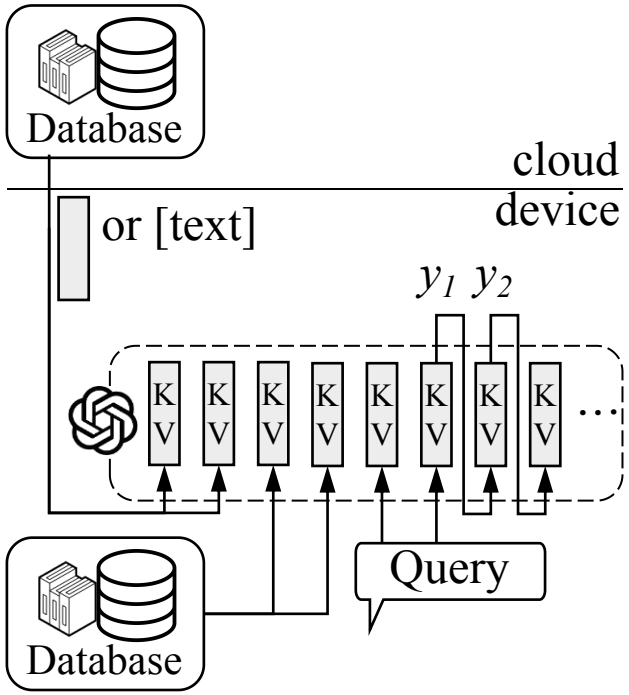
Distributed RAG (DRAG)



A Dilemma

KV-Cache is widely employed in Transformers

KV-Cache is commonly **>1000,000 times larger** than the raw text input



Case 1:
Download pre-built KV
Data transmission latency

Case 2:
Download text + recompute KV
Computing latency

How to resolve this dilemma?

Design of DRAGON

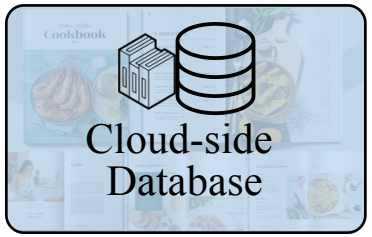
Design: A Rough Solution

Vanilla DRAG (VDRAG) – Workflow Decomposition

$$\underbrace{p(x_t|x_{<t})}_{\text{predict next-token using current context}} = \sum_d \overbrace{p(d|x_{<t})}^{\text{probability to retrieve document } d \text{ given current context}} \cdot \underbrace{p(x_t|d,x_{<t})}_{\text{predict next-token using enlarged context}} =$$

$$\sum_d p(d|x_{<t}) \cdot p(x_t|d,x_{<t}) \quad \text{for } d \in$$

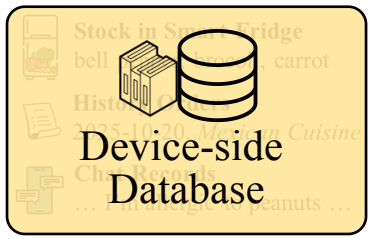
cloud-only RAG workflow $p^{\text{cloud}}(x_t|x_{<t})$



$+$

$$\sum_d p(d|x_{<t}) \cdot p(x_t|d,x_{<t}) \quad \text{for } d \in$$

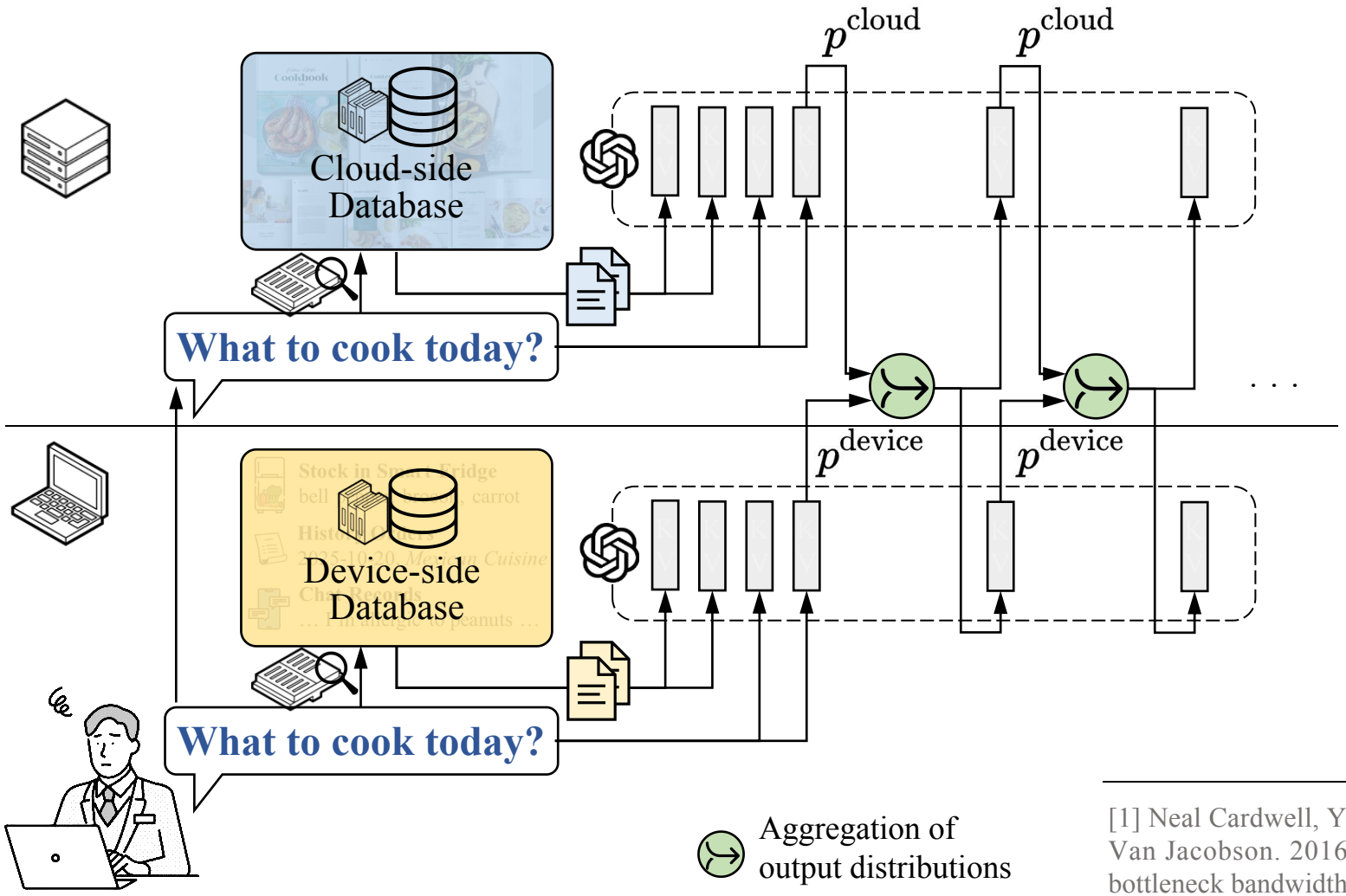
device-only RAG workflow $p^{\text{device}}(x_t|x_{<t})$



DRAG transformed into **output aggregations** between cloud-side RAG and device-side RAG

Design: A Rough Solution

Vanilla DRAG (VDRAG) – Workflow Decomposition



Predicting current token **requires waiting** for both output distributions to be generated, collected, and aggregated

Due to **small data packet size**, transmission time is often dominated by data-independent factors[1], like RTT

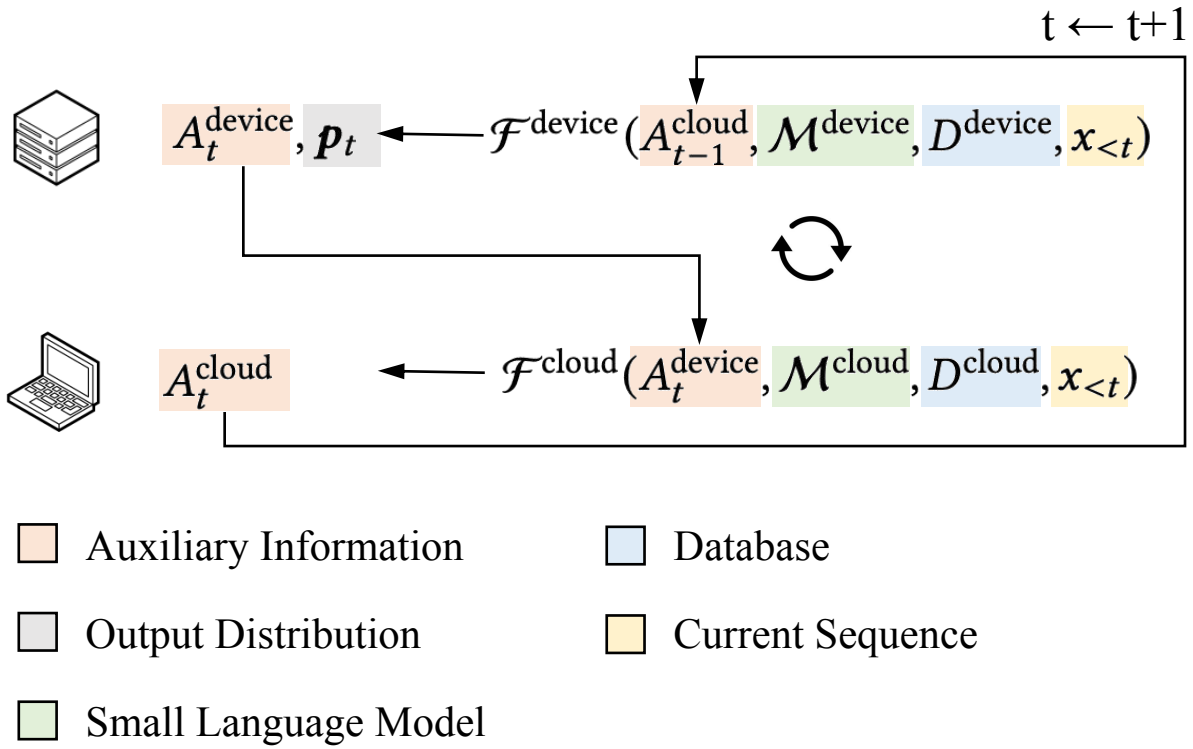
High-frequency synchronization reulted in **unacceptable transmission latency**

[1] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time. Queue 14 (2016), 20–53.

Design: Problem Formulation

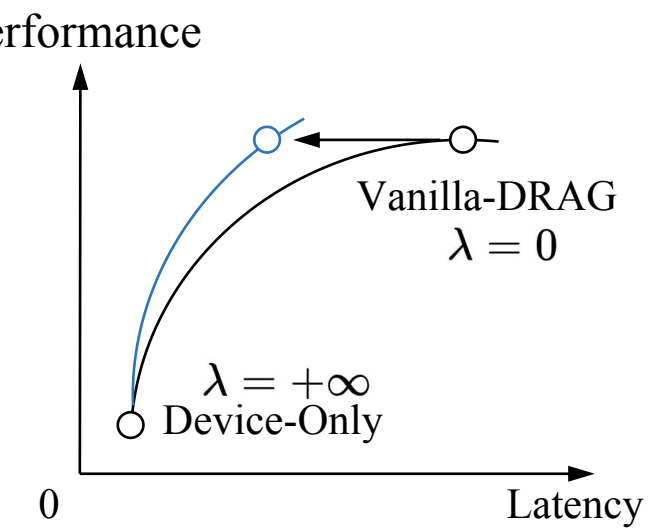
System Model

The device-side and cloud-side token generation processes $\mathcal{F}^{\text{device}}$ and $\mathcal{F}^{\text{cloud}}$ executes iteratively



Objective Function

$$\min_{\mathcal{F}} \frac{1}{N} \sum_{t=1}^{N} \underbrace{\left(-p(x_t^*|x_{<t}) \log \boldsymbol{p}_t(x_t^*|x_{<t})\right)}_{\text{cross-entropy loss}} + \underbrace{\lambda C(A_t, \mathcal{F})}_{\text{Latency}}$$



Push the Pareto frontier: minimize latency without compromising performance via proper **design of $\mathcal{F}$ / A**

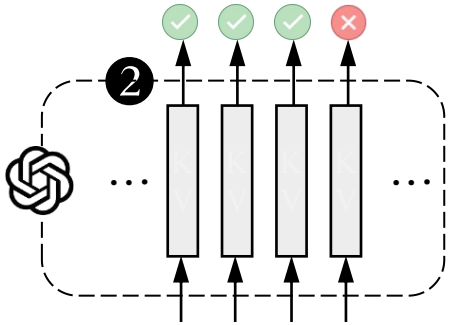
Design: Speculative Aggregation

Speculative Sampling

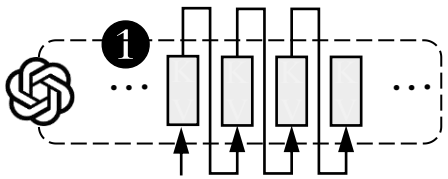
Draft-then-verify
Paralleled decoding
Reduce the number of calls

Insight

Some tokens can be speculated by a lightweight draft model



Target Model
verifies draft tokens in parallel



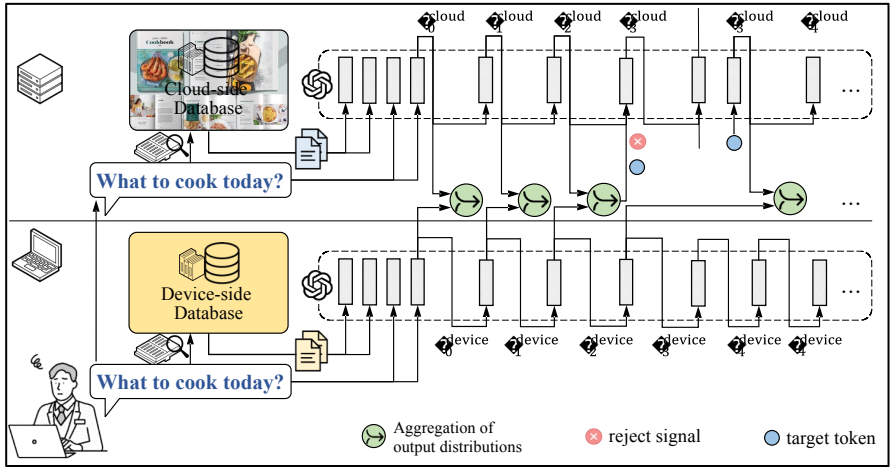
Draft Model
generates multiple tokens

Speculative Aggregation

Break the sequential dependence in VDRAG
Reduce the number of aggregations between two sides.

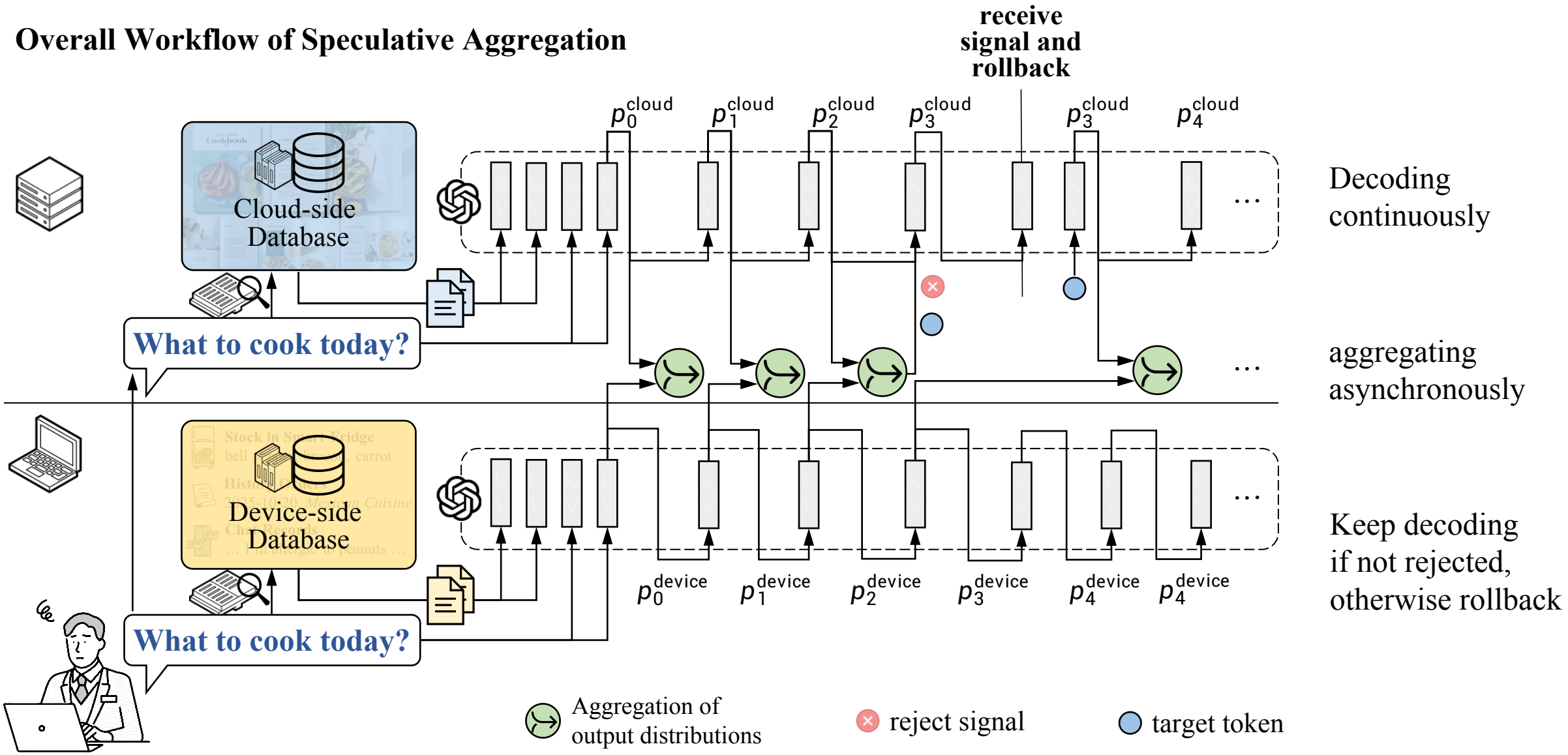
Insight

context-independent tokens should be similar across the device and the cloud



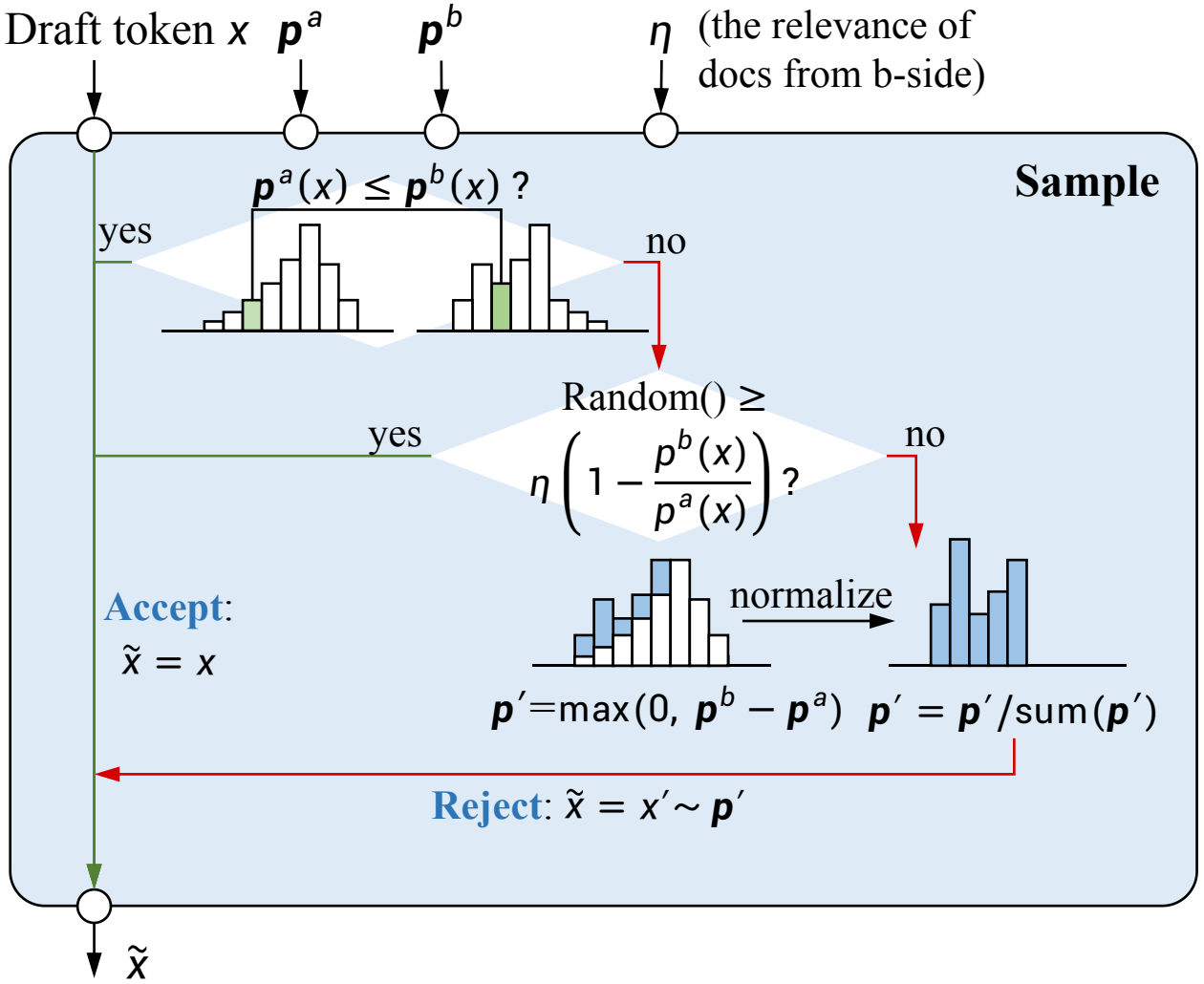
Design: Speculative Aggregation

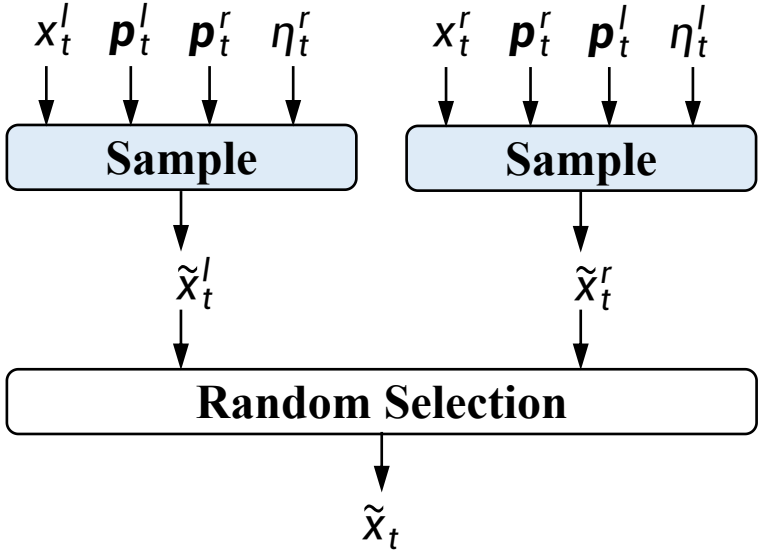
Overall Workflow of Speculative Aggregation



Design: Speculative Aggregation

Aggregation Strategy





Theorem 1

Target tokens produced by the speculative aggregation follows a distribution **identical** to that output by VDRAG.

Design: Greedy Scheduling

To **maximize the overlap** between decoding and transmission processes, adaptively schedule **which side performs the next aggregation**

Enumerate each acceptance scenario by assuming it **repeats continuously**

Dynamic Factors

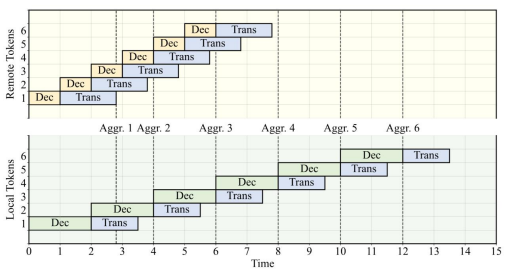
a **Acceptance Rates**

c_{dec} **Decoding Latency**

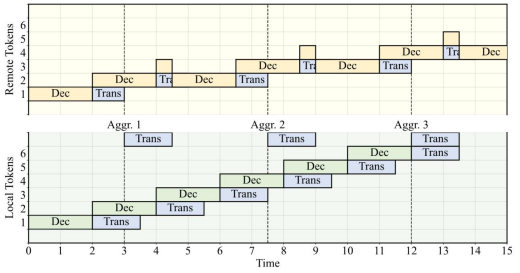
c_{trans} **Transmission Latency**

Estimation Model

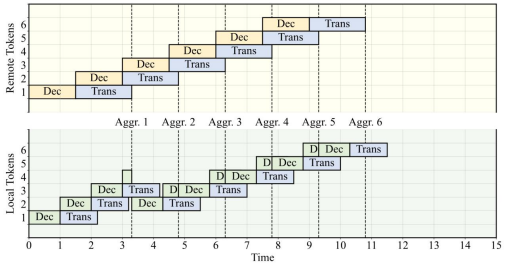
We employ a greedy strategy, where at each step, we **minimize the expected latency per token based on current observations**



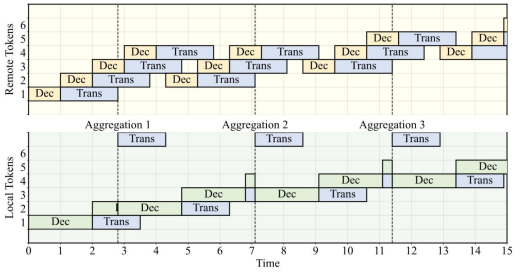
accept both



accept local, reject remote



reject local, accept remote



reject both

Design: Greedy Scheduling

Waiting time for each case

x_{t-1}^l	x_{t-1}^r	Waiting Time for x_t^l and x_t^r
rejected	accepted	$\max(c_{\text{dec}}^l, \varphi(c_{\text{dec}}^r + c_{\text{trans}}^r))$
accepted	rejected	$\max(\varphi(c_{\text{dec}}^l), c_{\text{trans}}^l + c_{\text{dec}}^r + c_{\text{trans}}^r)$
accepted	accepted	$\max(\varphi(c_{\text{dec}}^l), \varphi(c_{\text{dec}}^r + c_{\text{trans}}^r))$
rejected	rejected	$\max(c_{\text{dec}}^l, c_{\text{trans}}^l + c_{\text{trans}}^r + c_{\text{dec}}^r)$

Here, $\phi(T_{total}(task)) = \max(0, T_{total}(task) - T_{begin}(task) - T_{now}(task))$ means the remaining time of the given task

Probability of each case

We assume each case occurs with an expected probability given by the acceptance rate, e.g.,

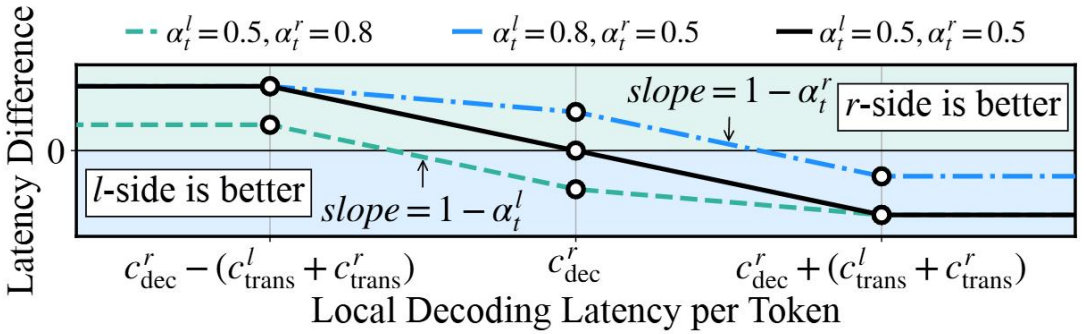
$$P(rejected) = E_{x\sim p_t^l(x)}(1 - \min(1, \eta_t^l + \eta_t^r p_t^r(x)/p_t^l(x))) = \eta_t^r \sum (p_t^l(x) - \min(p_t^l(x), p_t^r(x))).$$

Expected waiting time

$$Z = \alpha_t^r \max(c_{\text{dec}}^l, c_{\text{dec}}^r) + (1 - \alpha_t^r) \max(c_{\text{dec}}^l, c_{\text{dec}}^r + c_{\text{trans}}^l + c_{\text{trans}}^l)$$

Extra waiting time after switching aggregation side

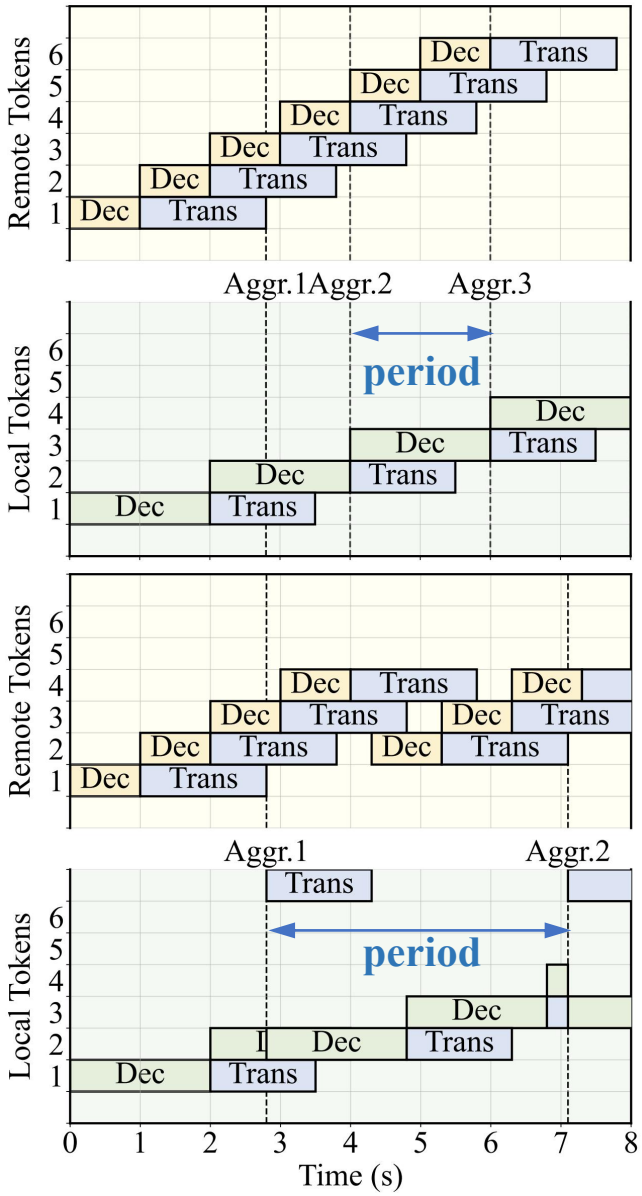
$$\Delta Z_t = \begin{cases} (1-\alpha_t^r)\text{rtt}, & c_{\text{dec}}^l \le c_{\text{dec}}^r - \text{rtt} \\ (1-\alpha_t^l)j + (\alpha_t^l - \alpha_t^r)\text{rtt}, & c_{\text{dec}}^r - \text{rtt} < c_{\text{dec}}^l \le c_{\text{dec}}^r \\ (1-\alpha_t^r)j + (\alpha_t^l - \alpha_t^r)\text{rtt}, & c_{\text{dec}}^r < c_{\text{dec}}^l \le c_{\text{dec}}^r + \text{rtt} \\ (a_t^l - 1)\text{rtt}, & c_{\text{dec}}^r + \text{rtt} < c_{\text{dec}}^l \end{cases}$$



Scheduling Strategy

Switch the aggregation to the other side once $\Delta Z_t > 0$

Design: Theoretical Analysis



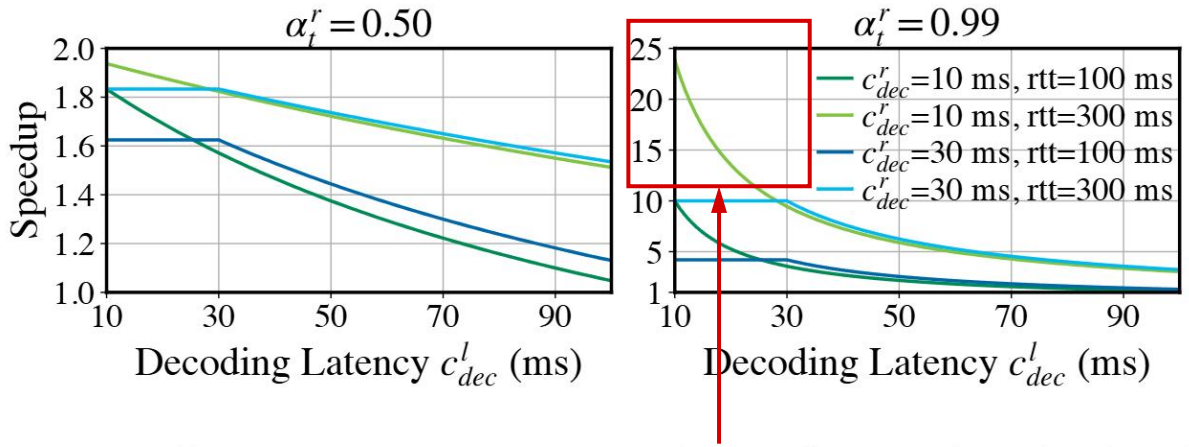
best case
Always accept

vs.

worst case
Always reject

The speedup is defined as $S_t = \tilde{Z}_t / Z_t$, where Z_t is the **expected waiting time** in greedy scheduling and $\tilde{Z}_t$ represents the worst-case latency

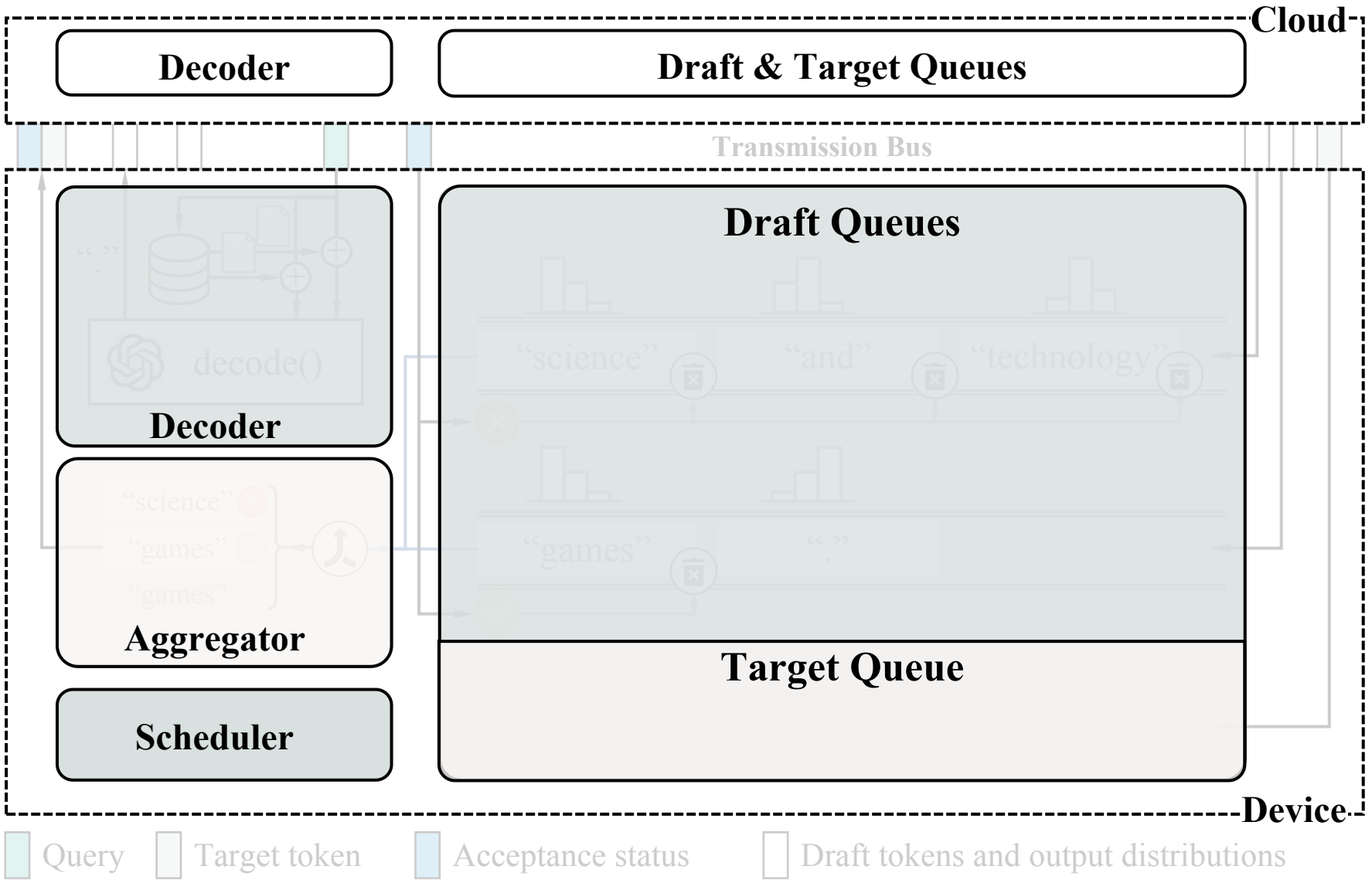
$$\frac{1}{S_t} = \begin{cases} 1 - \frac{\alpha_t^r}{1 + c_{dec}^r / rtt}, & c_{dec}^l \le c_{dec}^r, \\ 1 - (1 - \frac{c_{dec}^l}{c_{dec}^r + rtt})\alpha_t^r, & c_{dec}^r < c_{dec}^l \le c_{dec}^r + rtt, \\ 1, & c_{dec}^r + rtt < c_{dec}^l \end{cases}$$



Corollary 1. *DRAGON is particularly effective when the decoding latency gap between the device and the cloud is small and the transmission cost becomes the primary bottleneck.*

Corollary 2. *DRAGON's improvement in wall time can be substantially amplified when the cloud-side acceptance rate is high.*

Design: System Design

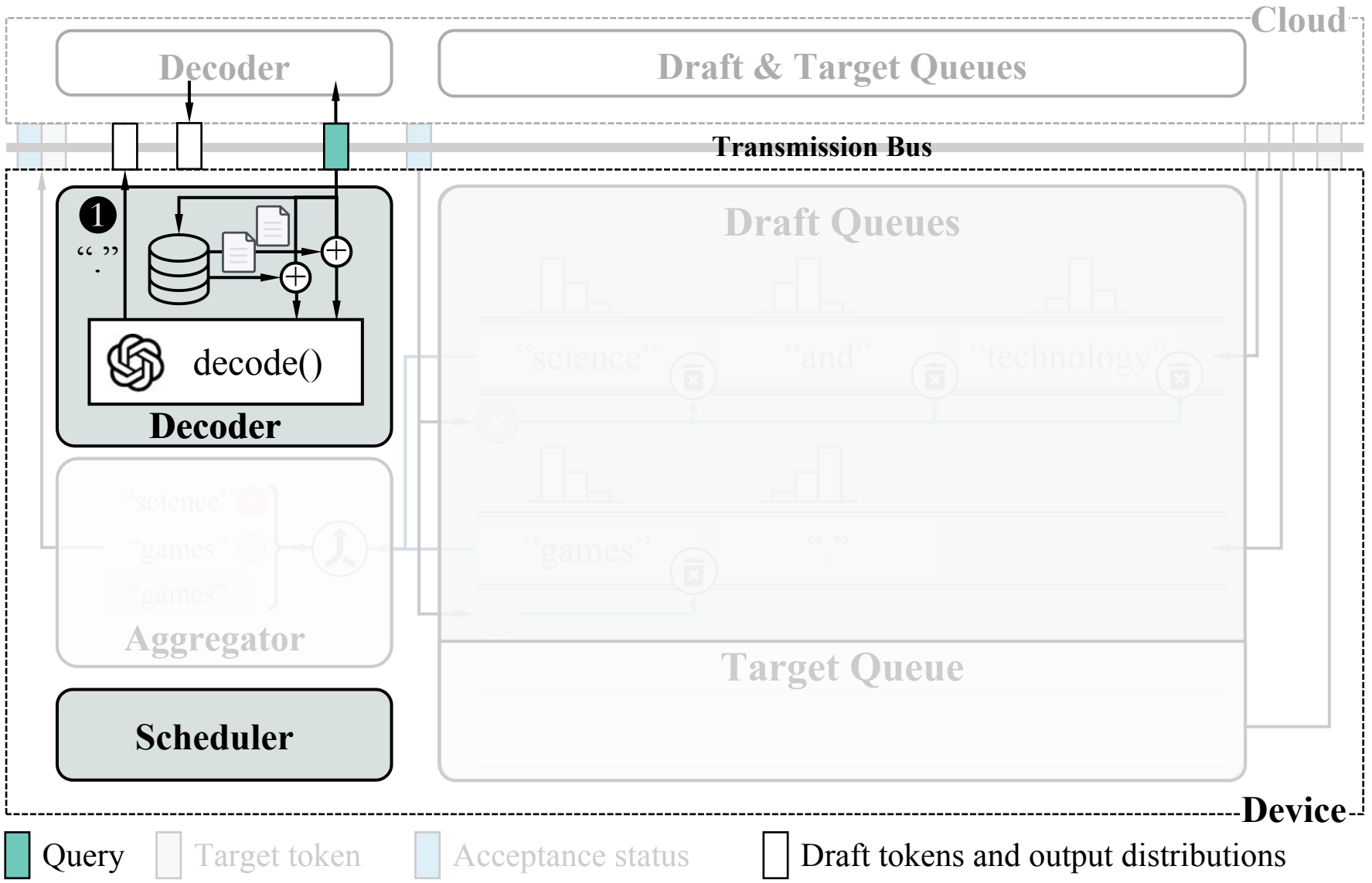


Overview of DRAGON

We propose DRAGON, a device-cloud distributed RAG framework.

We deploy Decoders, Queues, and Schedulers on both sides and an Aggregator module on either side

Design: System Design

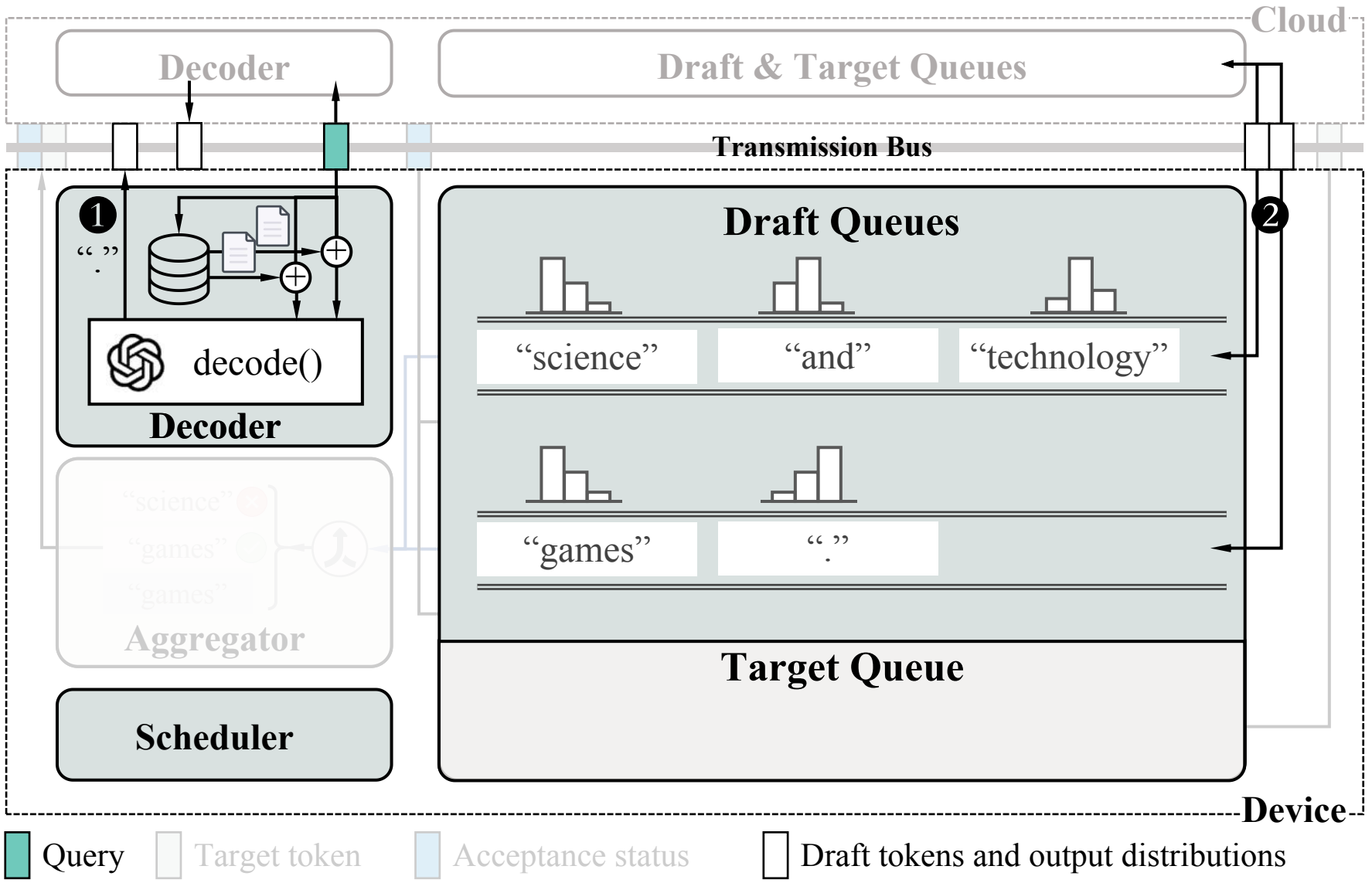


Overview of DRAGON

Step ❶

The Decoder module serves as a token producer, and on each side it decodes draft tokens independently based on local output distributions using the retrieved local documents only

Design: System Design

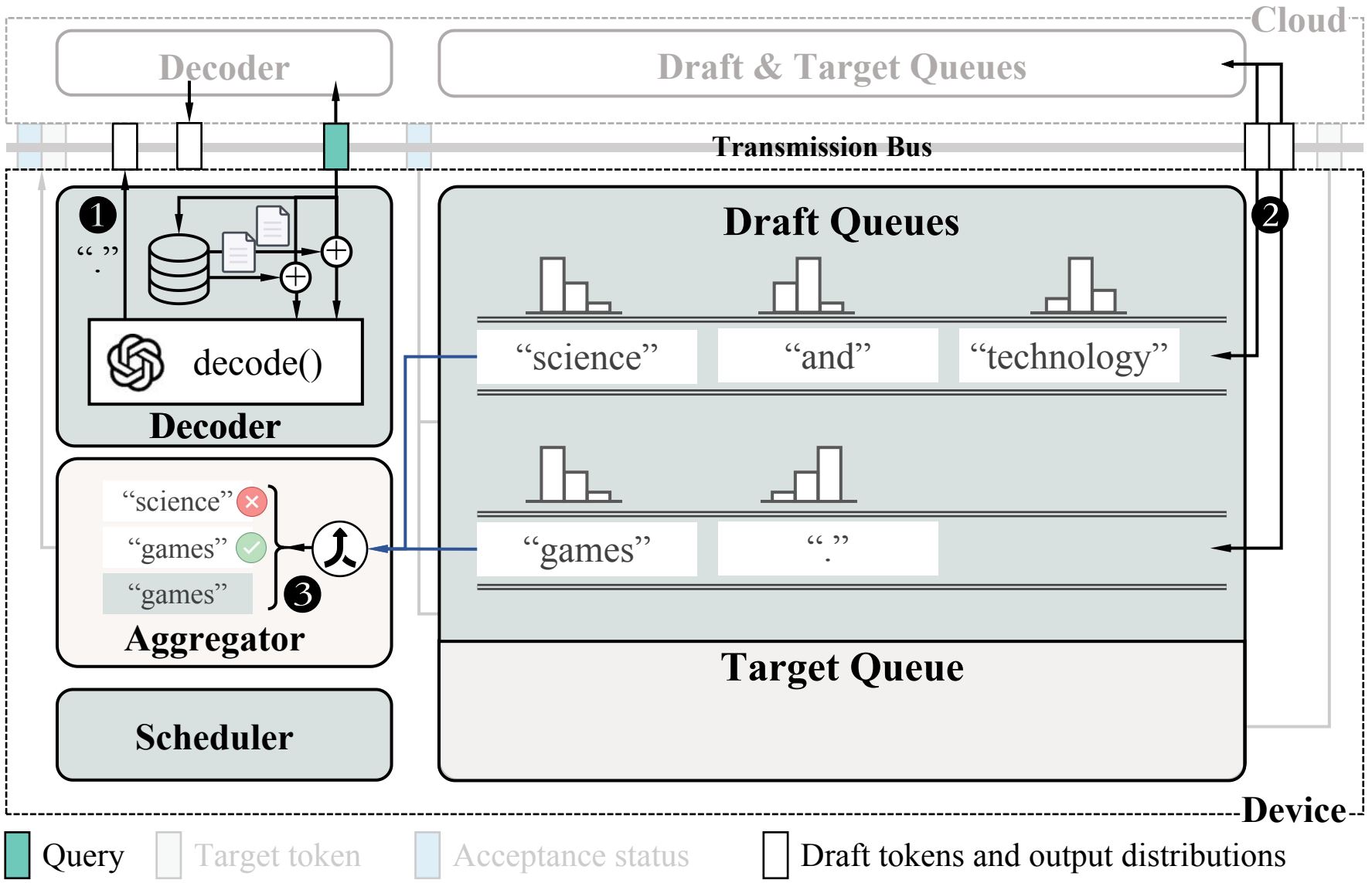


Overview of DRAGON

Step ❷

The draft tokens and their corresponding distribution vectors are broadcast to the other side. On each side, we enqueue into Draft Queues

Design: System Design

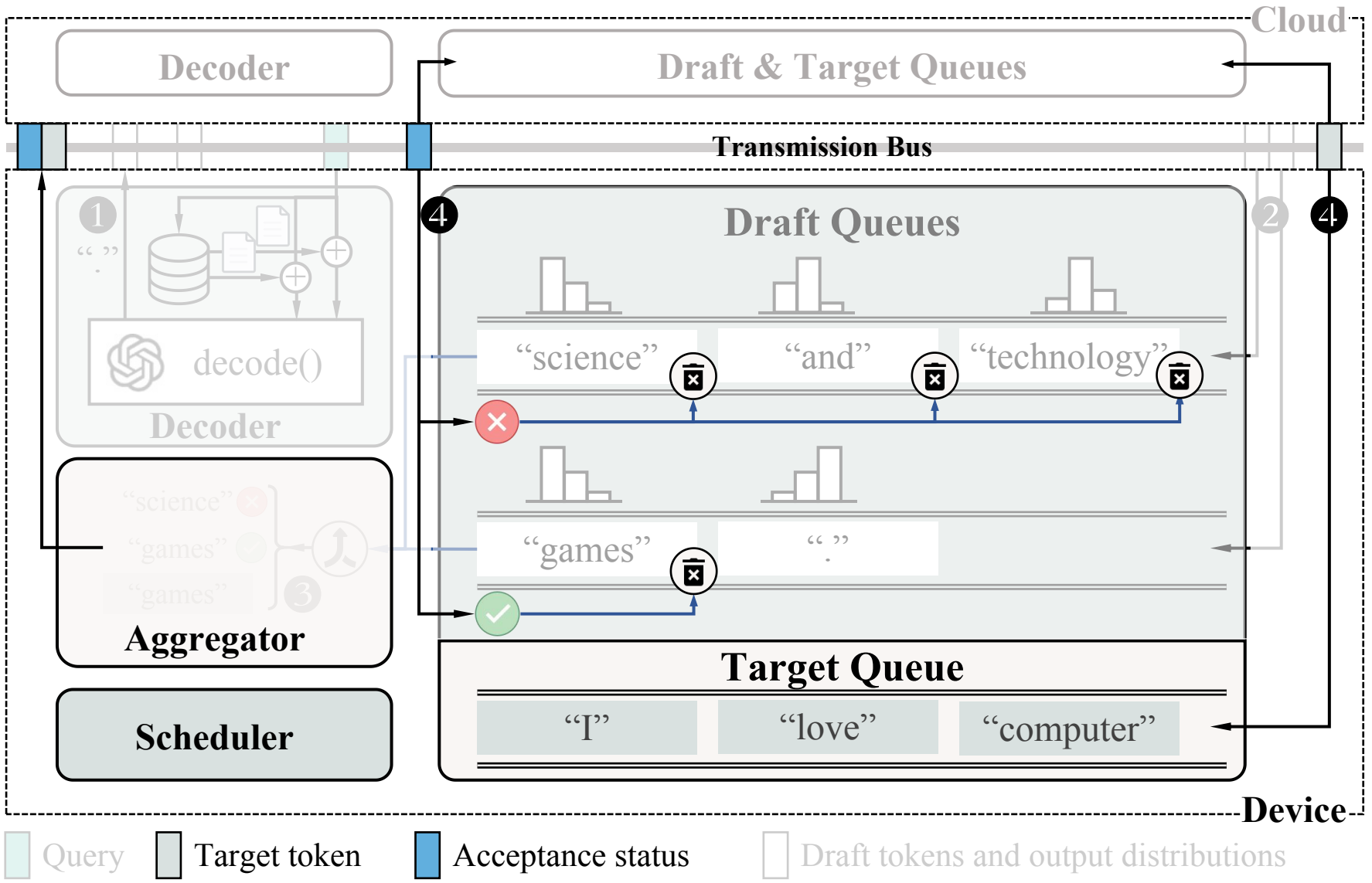


Overview of DRAGON

Step ❸

The Aggregator module, as a consumer, continuously consumes draft tokens from the front of local queues and performs aggregation process

Design: System Design



Overview of DRAGON

Step ❹

The aggregation results of the draft token are broadcast to Draft Queues on both sides.

For each queue, the first token is dequeued if accepted, or the entire queue is cleared if rejected. The final target token output by Aggregator is enqueued into Target Queue on both sides

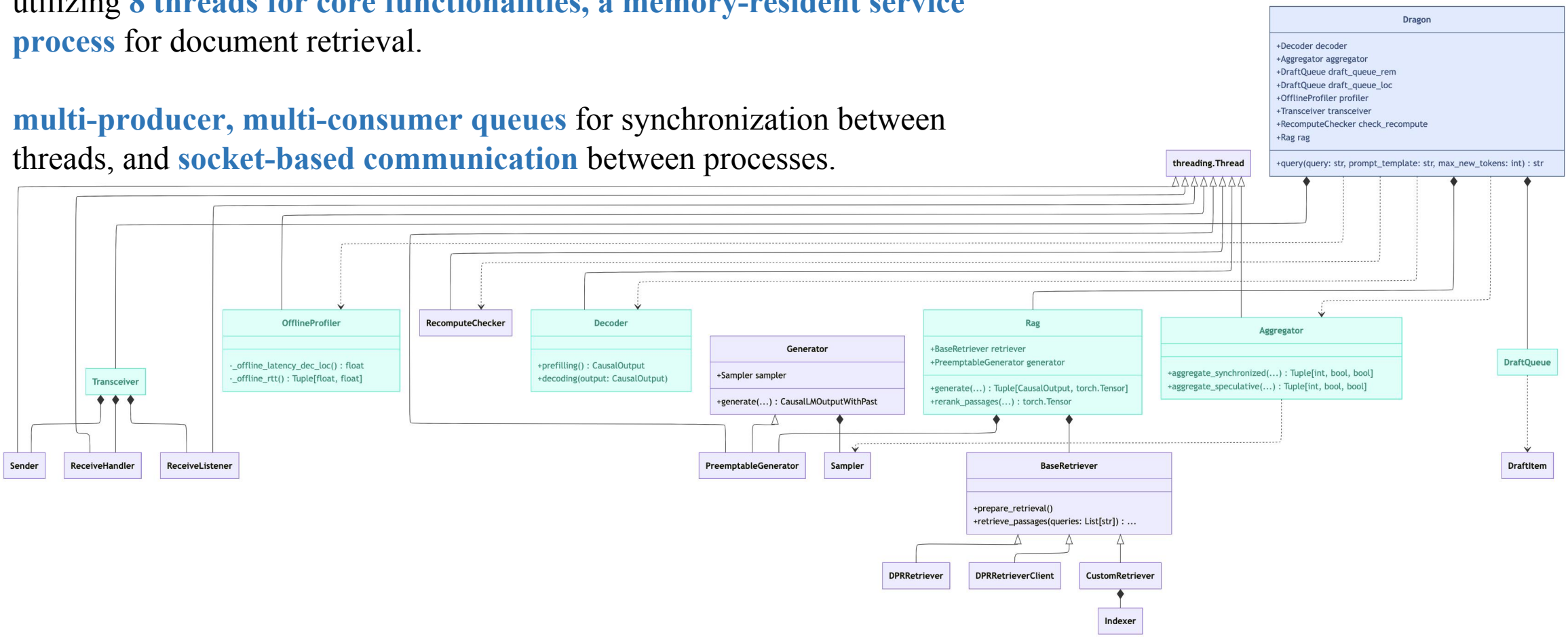
Experiments

Experiments: Implementation

We implemented DRAGON for distributed RAG workflow comprising ~3,000 lines of Python code: https://github.com/ThomasAtlantis/DRAGON

Two symmetric processes, the device-side and cloud-side ones, each utilizing **8 threads for core functionalities, a memory-resident service process** for document retrieval.

multi-producer, multi-consumer queues for synchronization between threads, and **socket-based communication** between processes.



Experiments: Implementation

Efficient transmission

- Pickle.dumps()/numpy.tobytes() → LZ4 compressor → struct.pack() → socket.send()
- For transmitting the output distributions $\boldsymbol{p}_t^{\text{device}}$ and $\boldsymbol{p}_t^{\text{cloud}}$, we employ **an aggressive top-p[1] selection strategy**
- Transmission data size is significantly **reduced by approximately 2,363×** when given the vocabulary size of 50,272, compared to the unoptimized JSON-based implementation.

Preemptible generation

We implemented **a hook function** to enable layer-wise interruption of draft decoding.

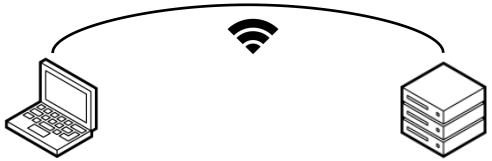
When interrupted, it **rolls back the KV-cache and attention masks** based on the number of generated target tokens so far and feeds the latest target token as input to trigger re-computation.

[1] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, Yejin Choi:
The Curious Case of Neural Text Degeneration. ICLR 2020

Experiments: Evaluation

Experiment Setups

Testbed 2.4 GHz Wi-Fi local-area network
Latency=2ms, Jitter=6ms



Node	MacBook Pro	high-performance computer
CPU	Intel Core i7	Intel Xeon Silver 4210R
Memory	16 GB	64 GB
GPU	No dedicated	1×GeForce RTX 3090

We use the traffic control tool, *tc,* to simulate network jitter

Models

- **SLMs:** OPT-1.3B[2] and Qwen2.5-1.5[3]
- **Retriever:** Contriever[4] (for Language Modeling) and DPR[5] (for latency measurement)
- **Re-ranker:** ms-marco-MiniLM-L6-v2[6]

Datasets and metrics

- **Performance:** we test language modeling on WikiText[1] (>100M tokens, two different-scale versions, **WikiText2 / WikiText103**) using **perplexity** (i.e. exp cross-entropy).
- **Efficiency:** we measure the **time to first token** (TTFT) and **per-token latency**. We used the retrieval corpus and index pre-built by Facebook from a **Wikipedia dump** dated December 20, 2018, which contains 21 million documents.

[1] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer Sentinel Mixture Models. In ICLR.
[2] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, et al. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL]
[3] Qwen Team. 2024. Qwen2.5 Technical Report. arXiv:2412.15115
[4] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, et al. 2021. Unsupervised Dense Information Retrieval with Contrastive Learning.
[5] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, et al. 2020. Dense Passage Retrieval for OpenDomain Question Answering. In EMNLP. 6769–6781.
[6] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In EMNLP. 3980–3990.

Experiments: Evaluation

Experiment Setups

Baselines

We compare DRAGON with four baseline methods:

- **CRCG**, **centralized generation** augmented with **centralized retrieval** (most existing RAG methods)
- **DRCG**, on-device **centralized generation** augmented with **distributed retrieval**
- **DRDG/TW**, distributed RAG using token-wise synchronization (**VDRAG**)
- **DRDG/SW**, distributed RAG using **sequence-wise synchronization**, i.e., one-time aggregation of the independently generated output sequences [1][2]

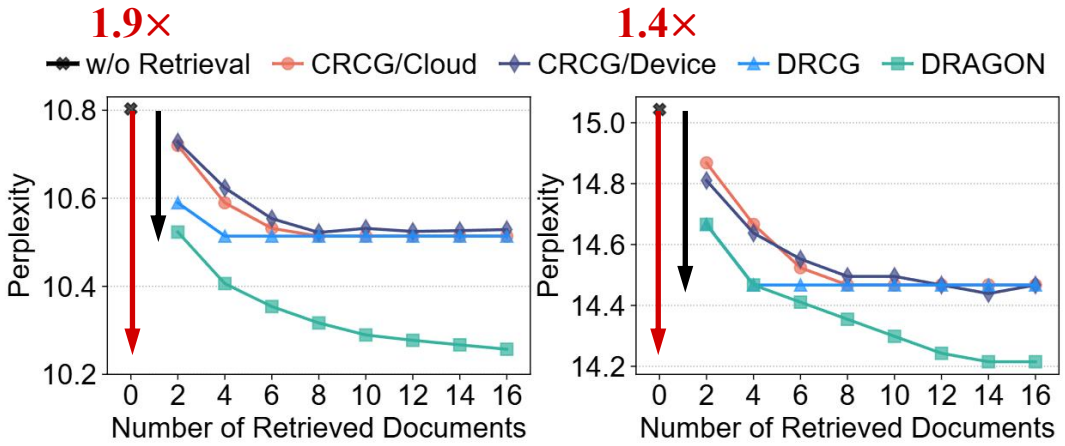
To study the overhead of DRCG, we evaluate two variants:

- **DRCG/Text** retrieves raw text and prefill KV cache from scratch
- **DRCG/KV** retrieves and reuses the KV cache of documents directly

[1] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2024. REPLUG: Retrieval-Augmented Black-Box Language Models. In NAACL. 8371–8384.
[2] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In NIPS. 9459–9474.

Experiments: Evaluation

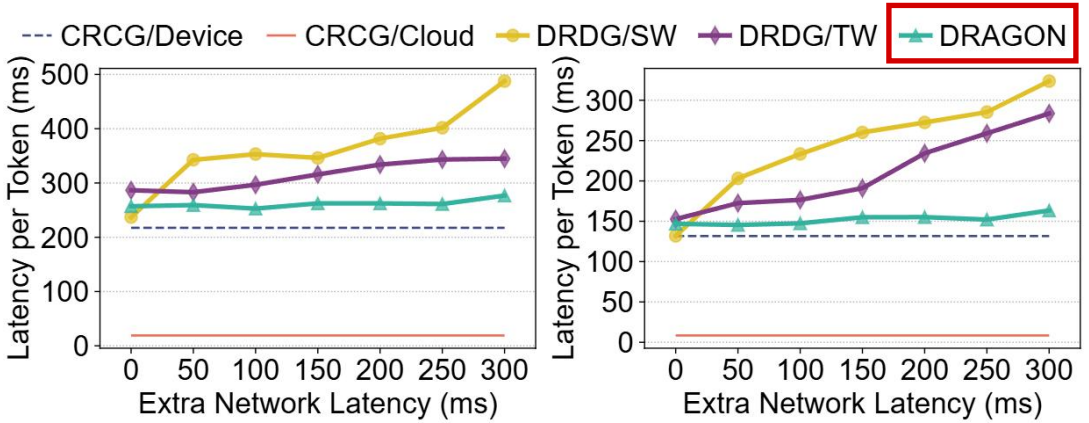
Overall Performance



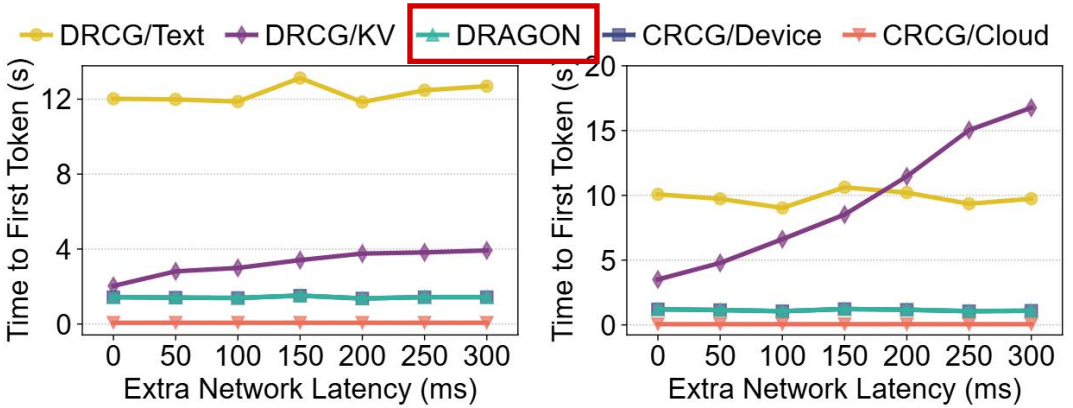
(a) Qwen2.5-1.5B/WikiText2 (b) OPT-1.3B/WikiText103
Performance on WikiText.

- DRAGON matches or outperforms all baseline methods across all settings.
- The performance gain increases as more documents are integrated.
- DRAGON shows strong robustness under different network conditions compared to distributed baselines

Overall Efficiency



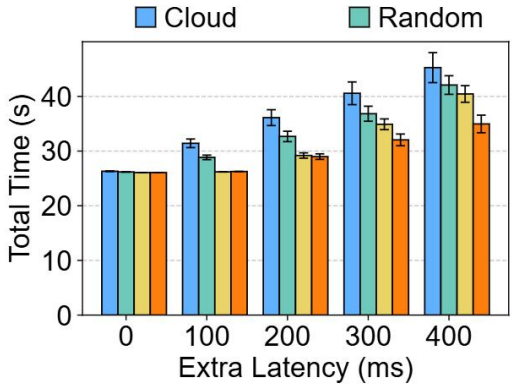
(a) Qwen2.5-1.5B (b) OPT-1.3B
Per-token latency in various network conditions.



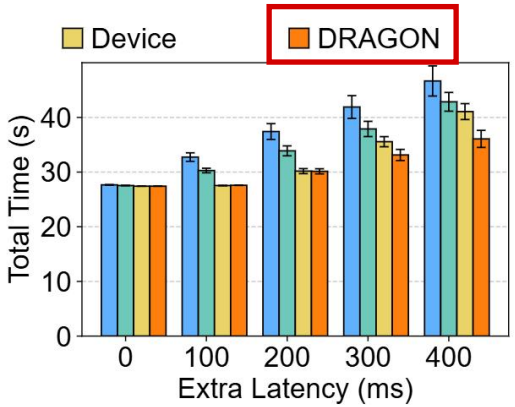
(a) Qwen2.5-1.5B (b) OPT-1.3B
Time-to-First-Token in various network conditions.

Experiments: Evaluation

Effectiveness of Scheduling



(a) Qwen2.5-1.5B

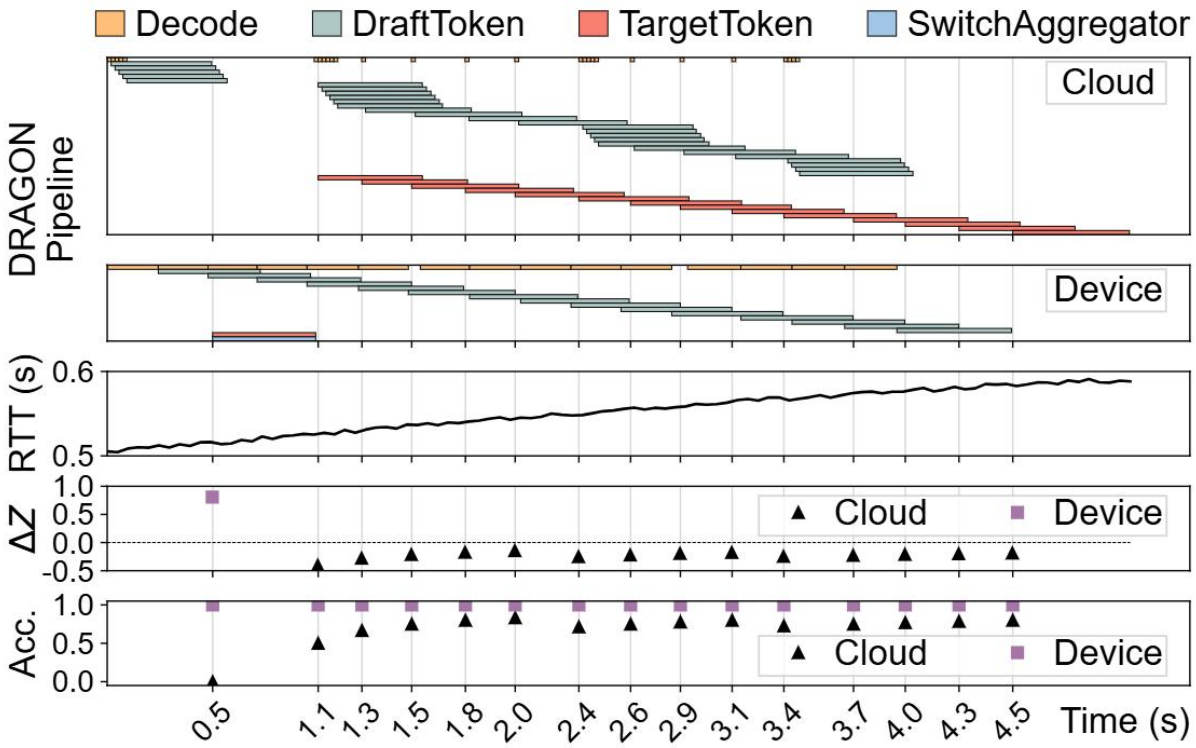


(b) OPT-1.3B

$C_{\text{dec}}^{\text{cloud}} < C_{\text{dec}}^{\text{device}} < C_{\text{dec}}^{\text{cloud}} + \text{rtt}$ consistently holds,

$$\Delta Z = (1 - a_t^{\text{cloud}})(C_{\text{dec}}^{\text{cloud}} - C_{\text{dec}}^{\text{device}}) + (a_t^{\text{device}} - a_t^{\text{cloud}})\text{rtt}$$

A B



T=0, by default
Aggregate
on the device side

T≥0.5, **B** dominates
Switch to aggregate
on the cloud side

Experiments: Evaluation

Overhead Analysis

Algorithm 1: SpeculativeAggregation

Input: Draft tokens x_t^s, locally-aggregated distributions $\boldsymbol{p}_t^s$, and aggregation weights h_t^s, for $s \in \{l, r\}$

Output: Target token x_t, acceptance status $\mathcal{S}^l$ and $\mathcal{S}^r$

Function Sample(x, $\boldsymbol{p}^a$, $\boldsymbol{p}^b$, η):

  $\tilde{x} \leftarrow x$, $\sigma^a \sim U(0,1)$;

  if $\boldsymbol{p}^a(x) > \boldsymbol{p}^b(x)$, $\sigma^a < \eta(1 - \boldsymbol{p}^b(x) \,/\, \boldsymbol{p}^a(x))$ **then**

    $\tilde{x} \sim \text{norm}(\max(0, \boldsymbol{p}^b - \boldsymbol{p}^a))$;

  return $\tilde{x}$;

$\eta_t^l \leftarrow h_t^l/(h_t^l + h_t^r)$, $\eta_t^r \leftarrow 1 - \eta_t^l$;

$\tilde{x}_t^l \leftarrow \text{Sample}(x_t^l, \boldsymbol{p}_t^l, \boldsymbol{p}_t^r, \eta_t^r)$, $\tilde{x}_t^r \leftarrow \text{Sample}(x_t^r, \boldsymbol{p}_t^r, \boldsymbol{p}_t^l, \eta_t^l)$;

$\sigma \sim U(0,1)$, $x_t \leftarrow \tilde{x}_t^l \cdot \mathbf{1}_{\sigma \le 0.5} + \tilde{x}_t^r \cdot \mathbf{1}_{\sigma > 0.5}$;

$\mathcal{S}^l \leftarrow x_t^l = x_t$, $\mathcal{S}^r \leftarrow x_t^r = x_t$;

return x_t, $\mathcal{S}^l$, $\mathcal{S}^r$;

Computational Overhead

An additional sampling operation at each decoding step

Scalar-Level
2 random generations, 1 subtractions
2 multiplications, 2 comparisons

Vector-Level
1 subtraction, 1 comparison, 1 normalization

For Qwen2.5-1.5B, the overhead is $< 3 \times 10^5$ MACs, accounting for 0.03% of the cost of decoding a single token.

Communication Overhead

Transmitting the output distributions incurs **less than 0.5 KB of extra data per token**. Under a 100 Mbps local-area Wi-Fi network, this translates to <40 us of latency.

Conclusion

Conclusion

- DRAGON is a distributed RAG framework that enhances on-device SLMs using both personal and general knowledge without raw document transmission.
- DRAGON partitions the RAG workflow across device and cloud, using **Speculative Aggregation** to minimize output synchronization overhead.
- Experimental results show that DRAGON notably improves generation quality while maintaining low latency.
- Designed for peer-to-peer computing nodes, also applicable for multi-node systems

DRAGON:
Enhancing On-Device Model Performance with Distributed Retrieval-Augmented Generation

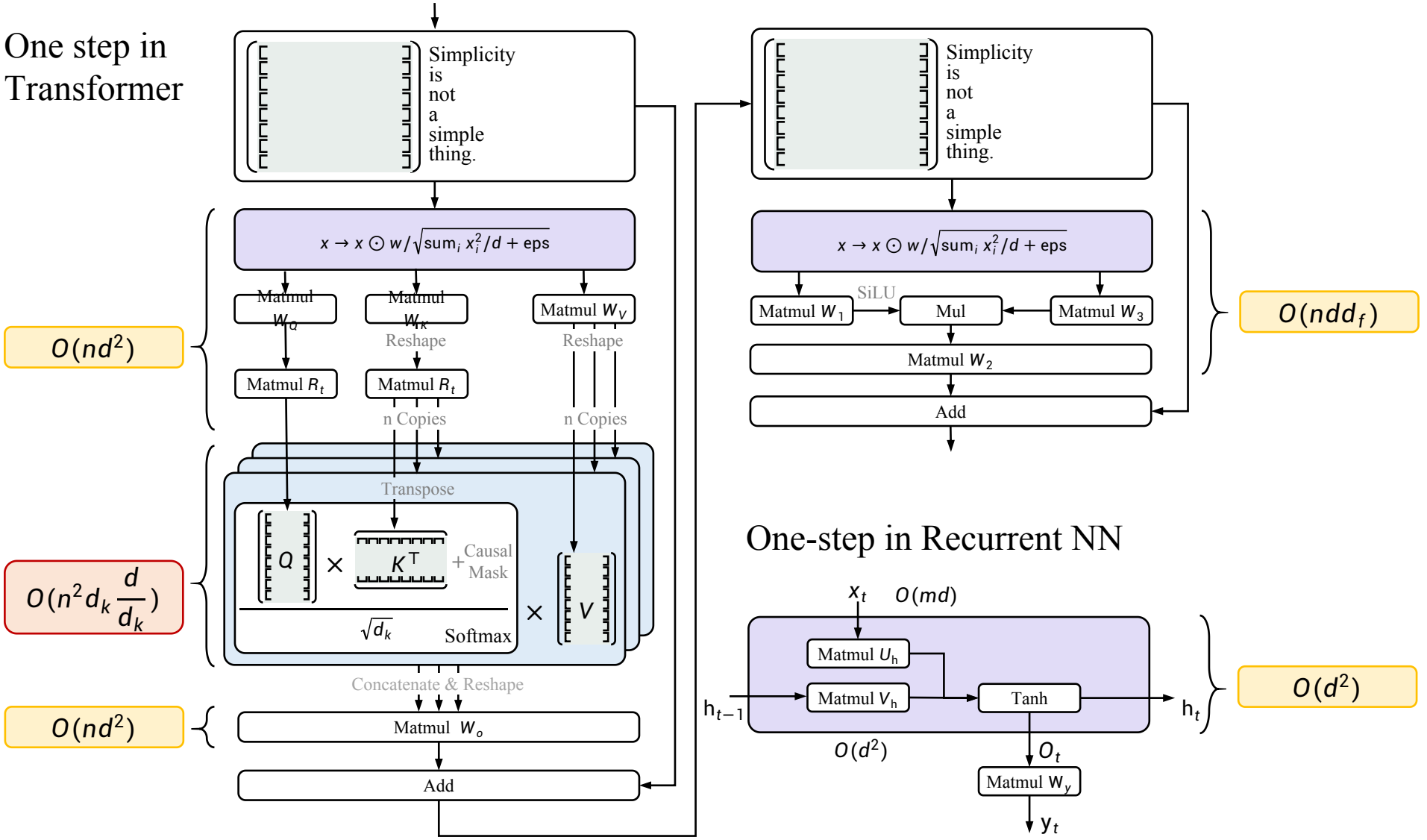
Thanks for your attention！ (Q&A)

Oct. 27th, 2025

ACM MobiHoc 2025, Houston, USA

Q&A: Why KV-Cache is employed in Transformer-based LLMs?

One step in Transformer

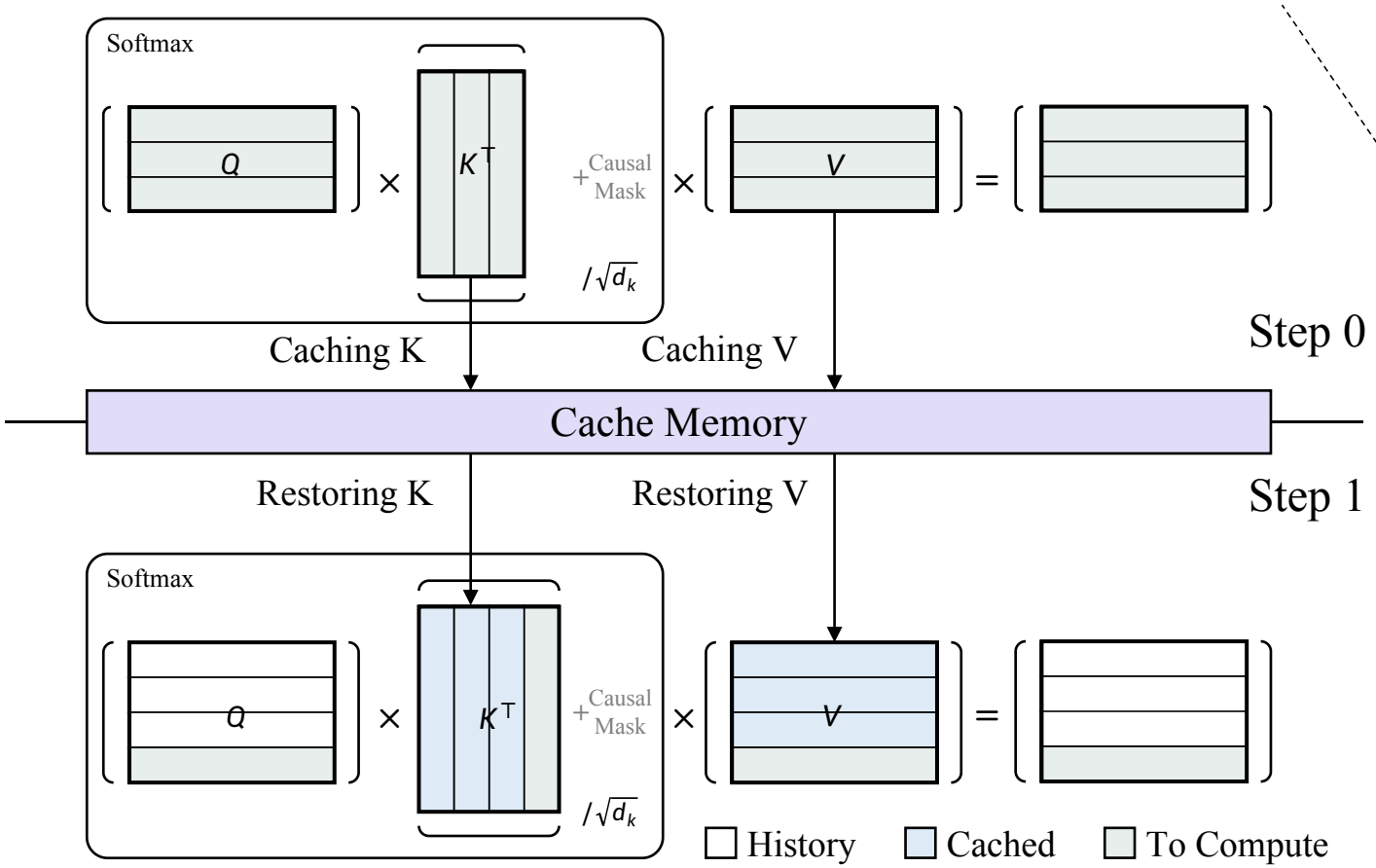


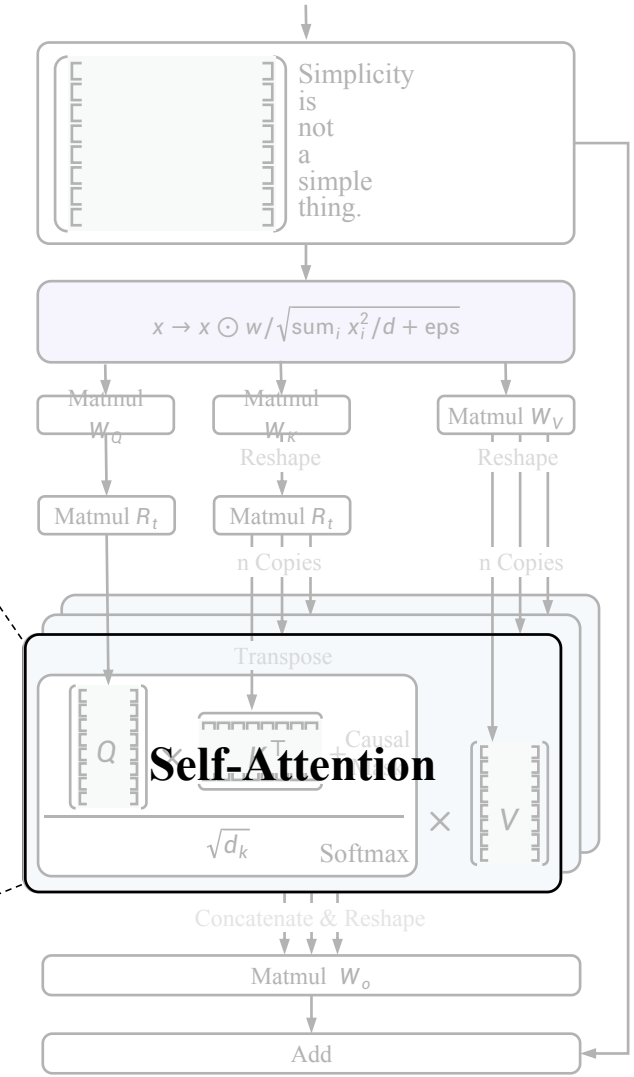
One-step in Recurrent NN

The relationship of self-attention is a **complete graph**, which results in a **quadratic** increase in time complexity as context length grows

Time complexity further exacerbated due to the enormous **hidden dimension** and the **number of layers**

KV Cache: Trade Space for Time





Each decoding step now operates with a time complexity of $O(nd + d^2)$

However, it incurs a space complexity of $O(nd)$.