HedraRAG: Co-Optimizing Generation and Retrieval for Heterogeneous RAG Workflows

Zhengding Hu[1], Vibha Murthy[1], Zaifeng Pan[1], Wanlu Li[2], Xiaoyi Fang[3], Yufei Ding[1], Yuke Wang[4]

1 Department of Computer Science & Engineering, UCSD

2 Nano and Chemical Engineering, UCSD

3 RegAilator Inc

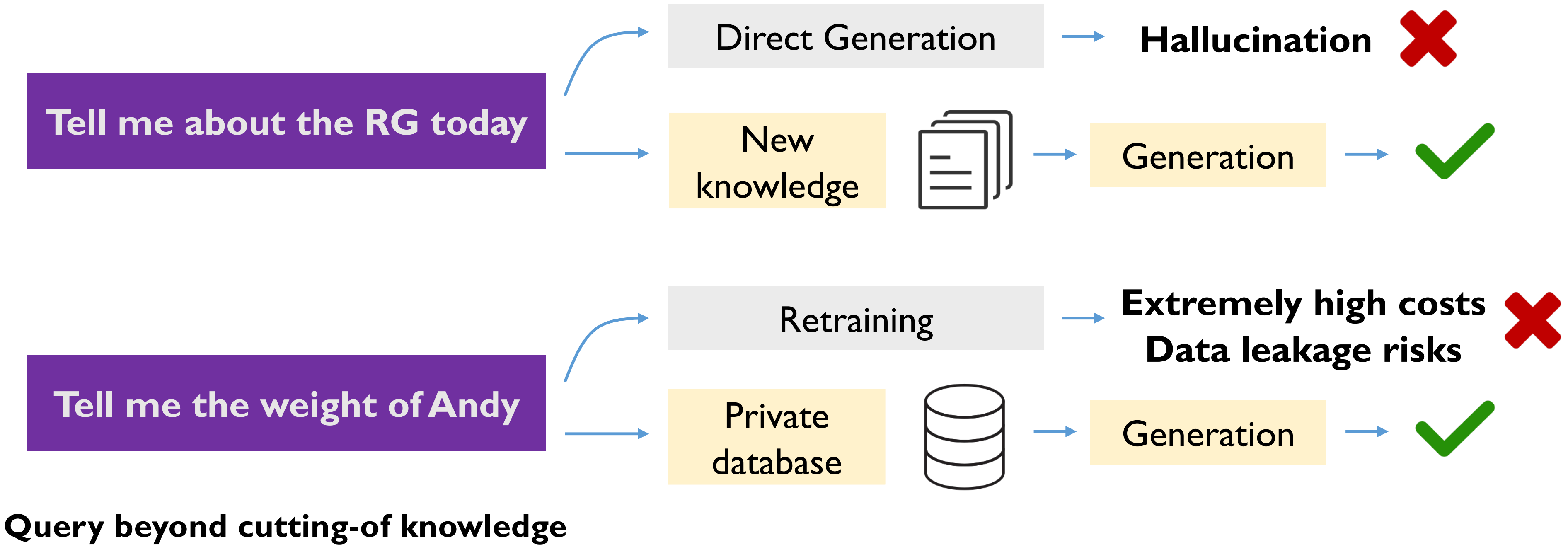
4 Computer Science, Rice University

Speaker: Chao Bi

SOSP' 25

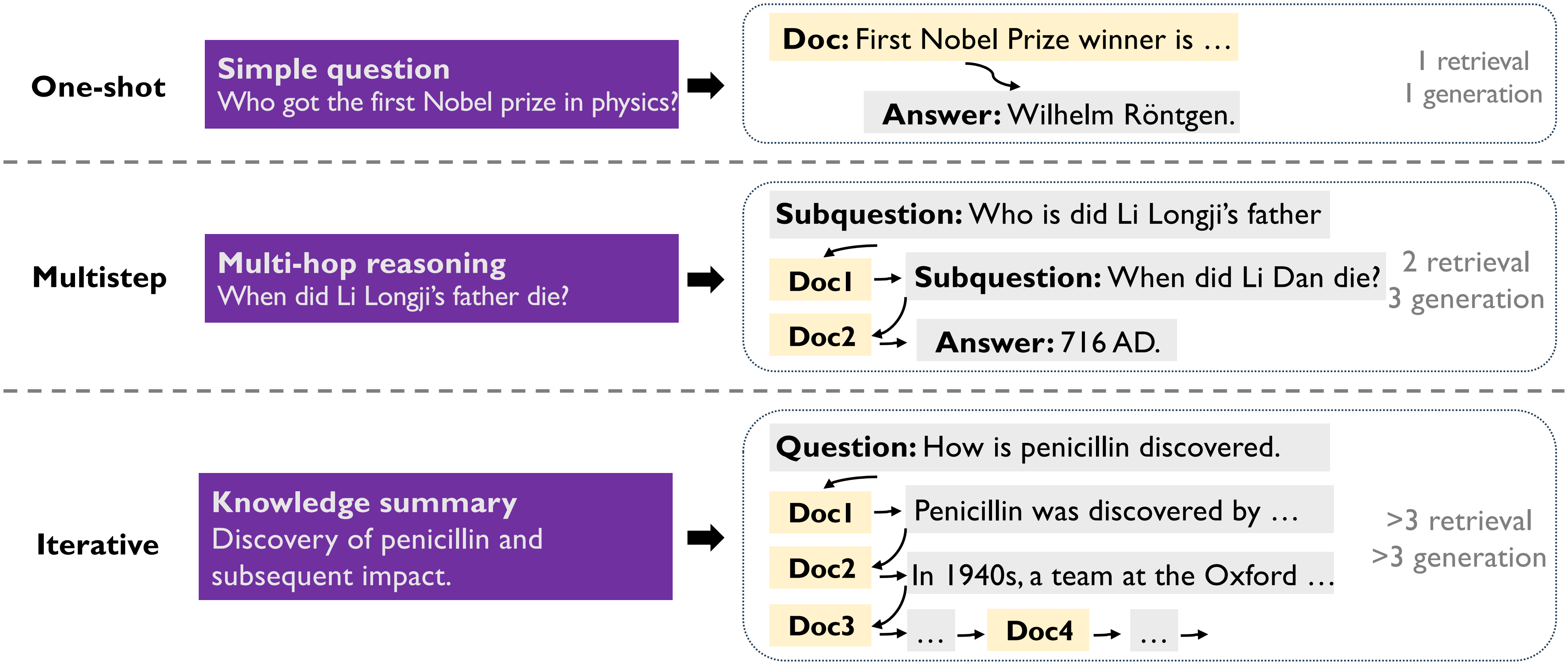
Retrieval-Augmented Generation (RAG)

- RAG is widely used in the era of LLM to
  - Avoid retraining, reduce hallucination, preserve data privacy



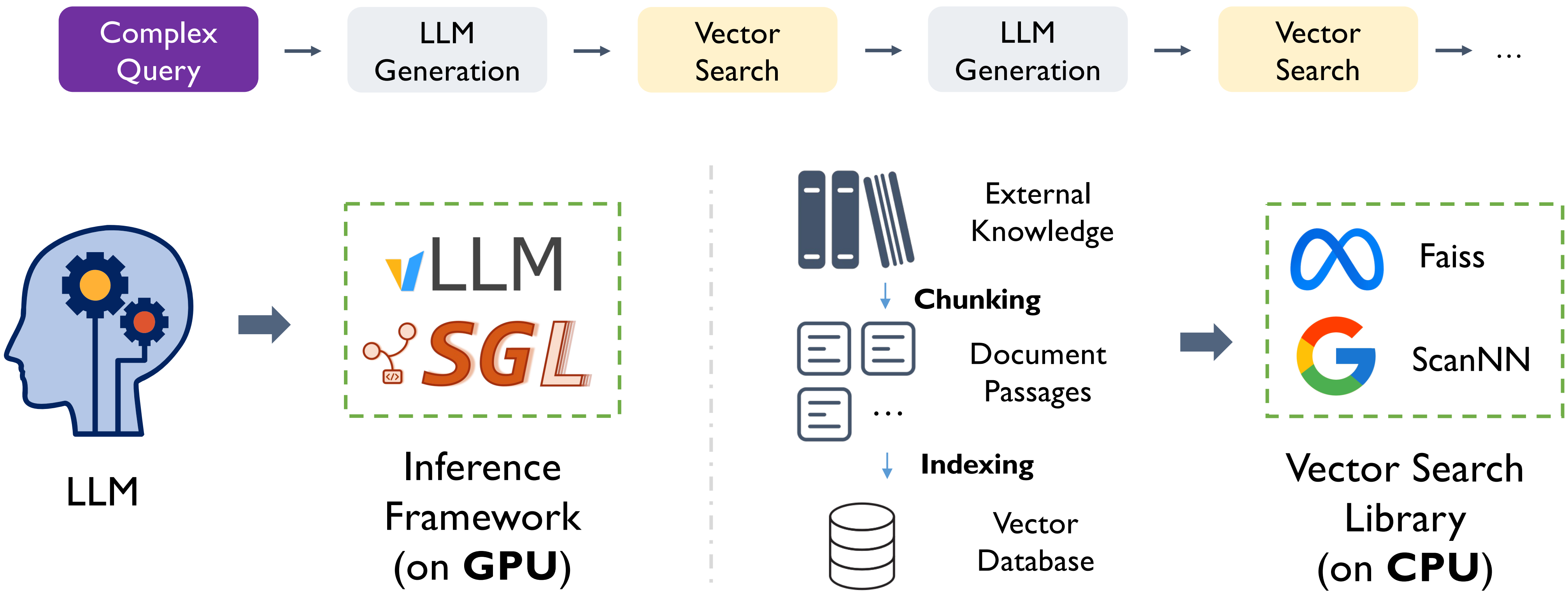
Heterogeneous RAG Workflow

- Complex LLM tasks drive RAG evolution



RAG Serving

- Iterative invocation between **Inference & Vector Search**



Motivation – Existing RAG Frameworks

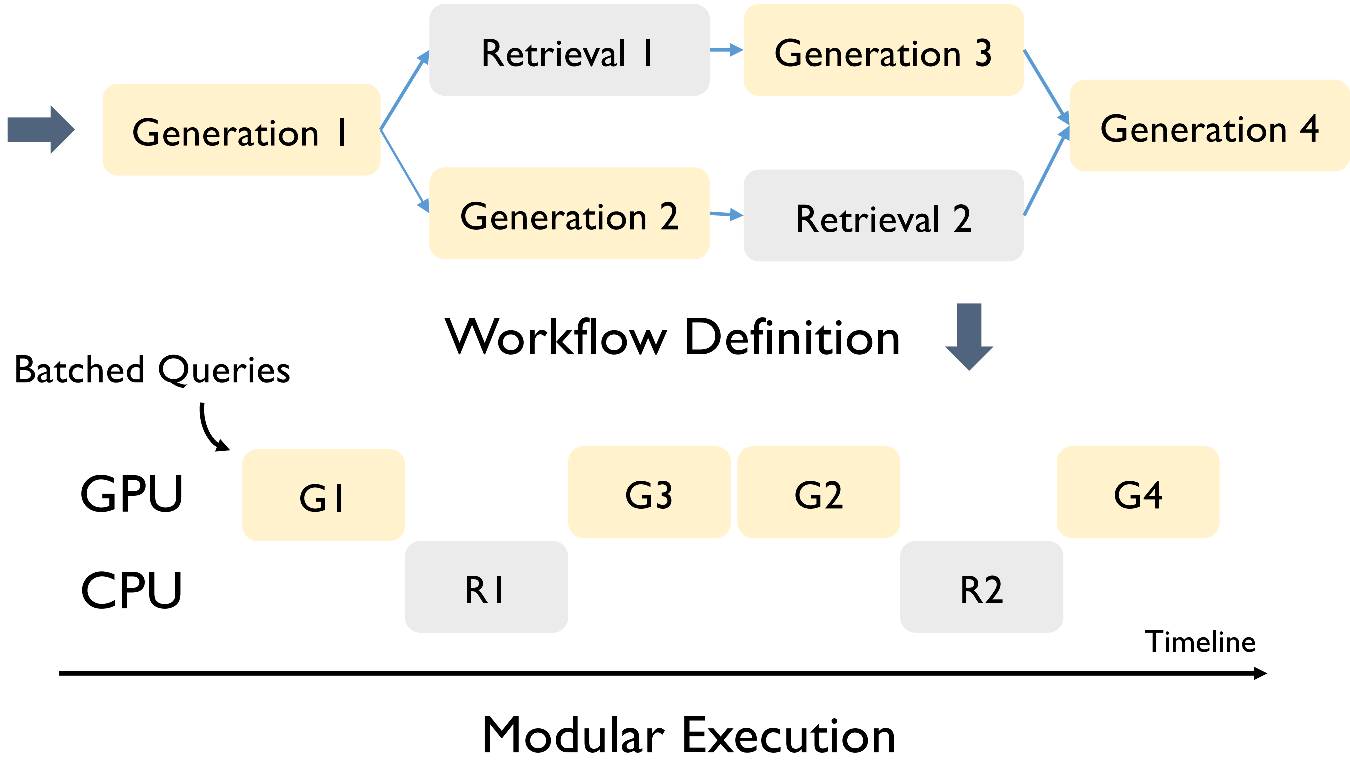
- **Functional** Emphasis: Modular, Synchronized Design

LlamaIndex

LangChain

n8n

RAG / Agentic Frameworks



Motivation – Limitations of existing solutions

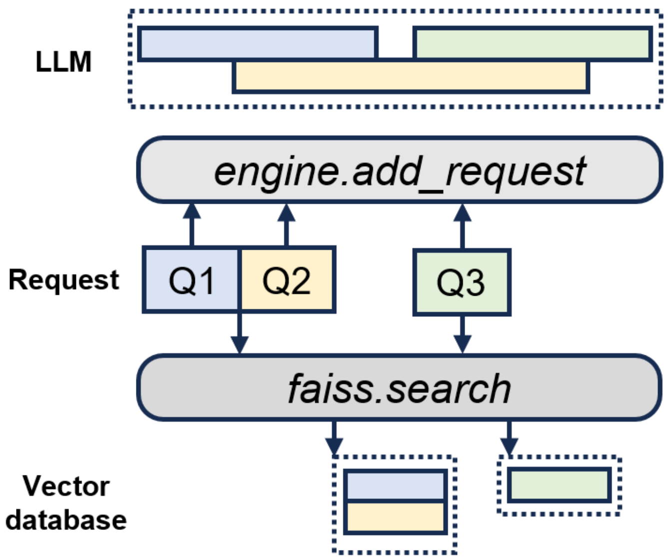
- **System Level**:
  - Focus on Resource scheduling and disaggregated deployment strategies[1, 2]
  - **Lack of unified runtime support** for coordinating multi-stage, heterogeneous flow

- **Algorithmic Level**:
  - Accelerating generation by reusing document prefixes[3, 4, 5]
  - Early-terminated retrieval[3] and speculative generation [6, 7]
  - **Missing generalizability** and sometimes **sacrificing output quality**

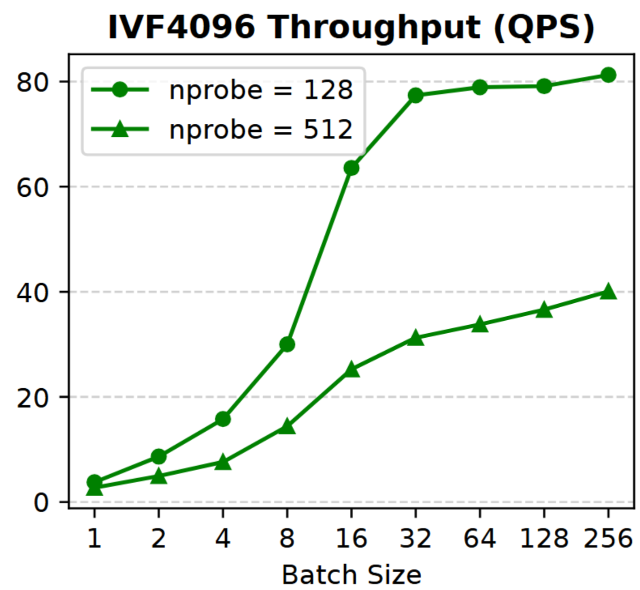
[1] RAGO: Systematic Performance Optimization for Retrieval-Augmented Generation Serving. (arXiv'25)
[2] Chameleon: a heterogeneous and disaggregated accelerator system for retrieval-augmented language models. (arXiv'23)
[3] RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation. (arXiv'24)
[4] Prompt cache: Modular attention reuse for low-latency inference. (MLSys'24)
[5] CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. (Eurosys'24)
[6] Piperag: Fast retrieval-augmented generation via algorithm-system co-design. (arXiv'24)
[7] Accelerating retrieval-augmented language model serving with speculation. (arXiv'24)

Opportunity – Stage-level parallelism

- **1st Observation**: gaps between LLM generation & vector search
  - LLM decoding: **Continuous** batching → Flexibly add new requests
  - Vector search: **Fixed** batching → strongly depends on batch size



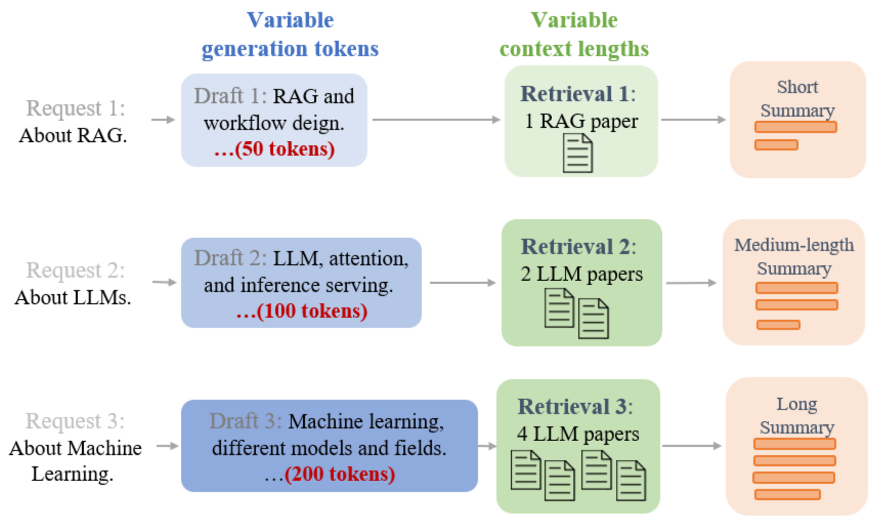
Batching Strategies



Throughput of vector search

Opportunity – Stage-level parallelism

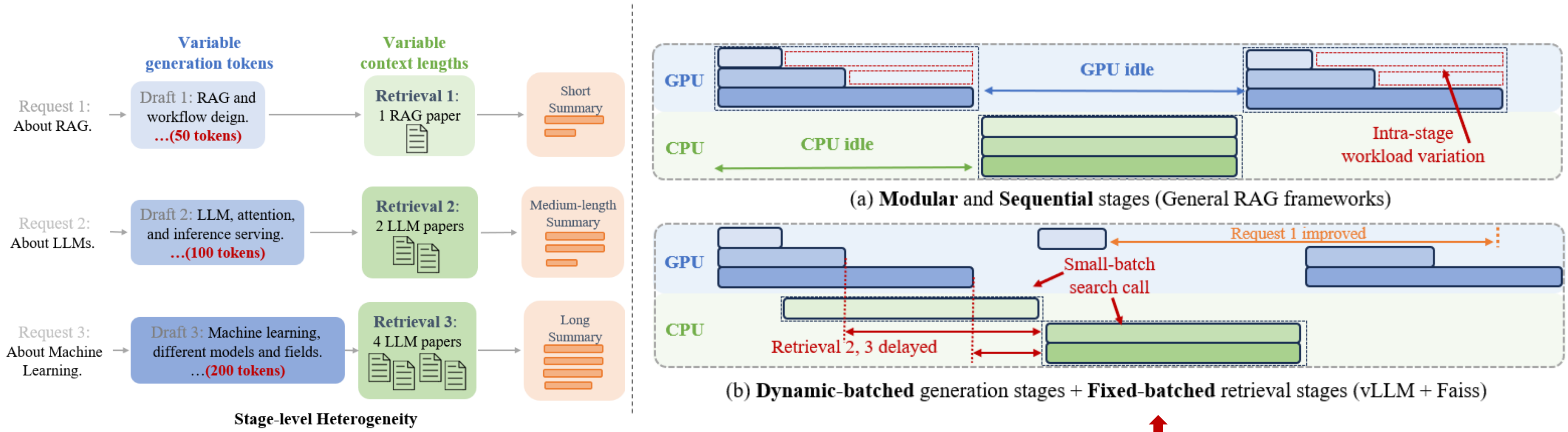
- Example: 3 request with **variable** generation lengths



Stage-level Heterogeneity

Opportunity – Stage-level parallelism

- Example: 3 request with **variable** Generation/Retrieval costs

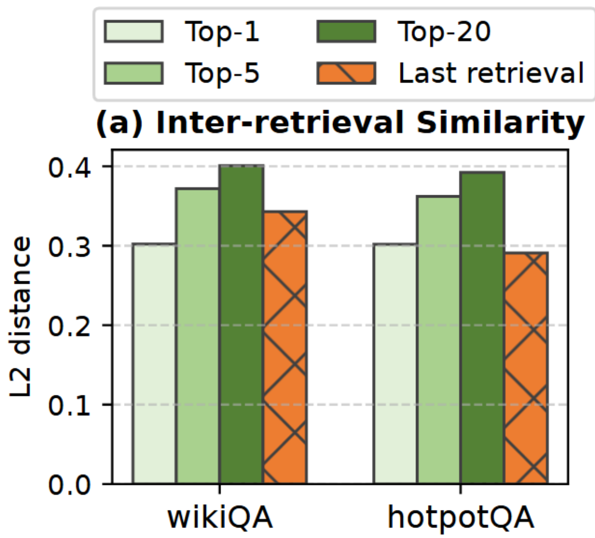


Stage-level Batching & Scheduling: CPU/GPU under-utilization

Opportunity – Intra-Request Semantic Similarity

- **2nd Observation:** Similarity between stages in iterative workflows

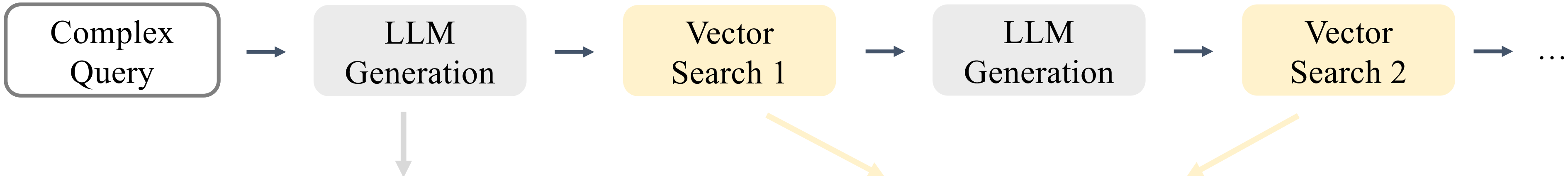
Complex Query → LLM Generation → Vector Search 1 → LLM Generation → Vector Search 2 → …

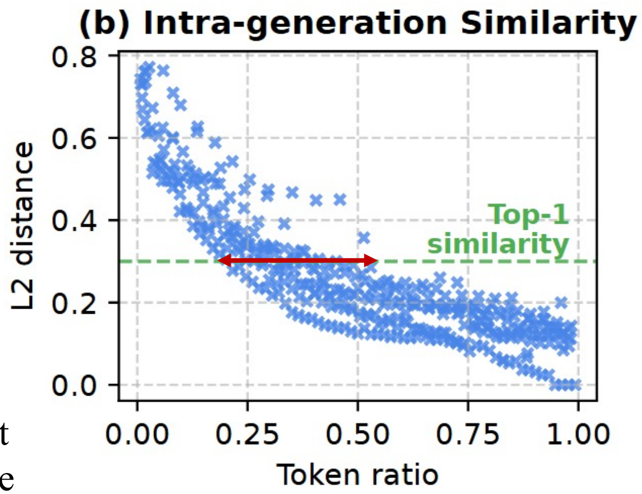


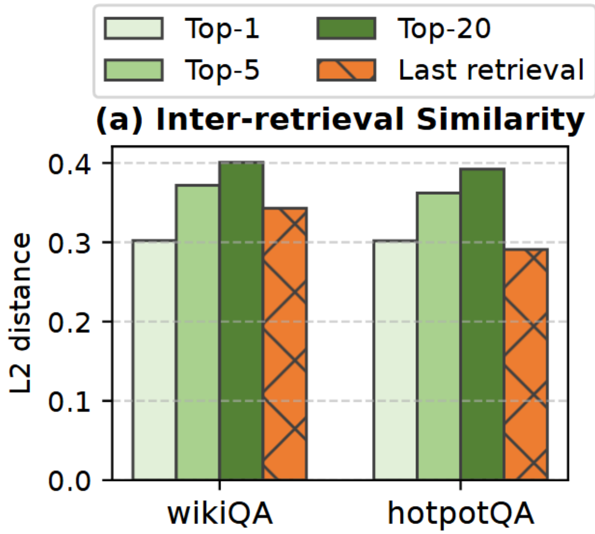
Successive retrievals yield **Close Vectors** in the embedding space

Opportunity – Intra-Request Semantic Similarity

- **2nd Observation**: Similarity between stages in iterative workflows



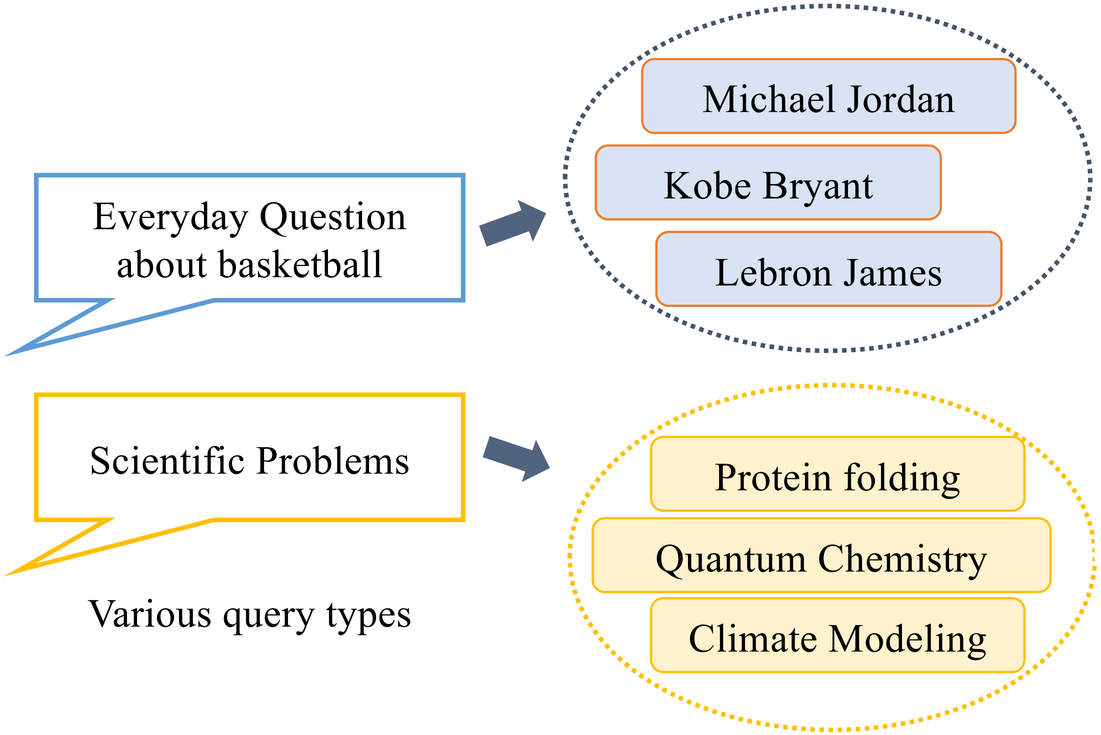




Partial generation is close to the final result in the embedding space

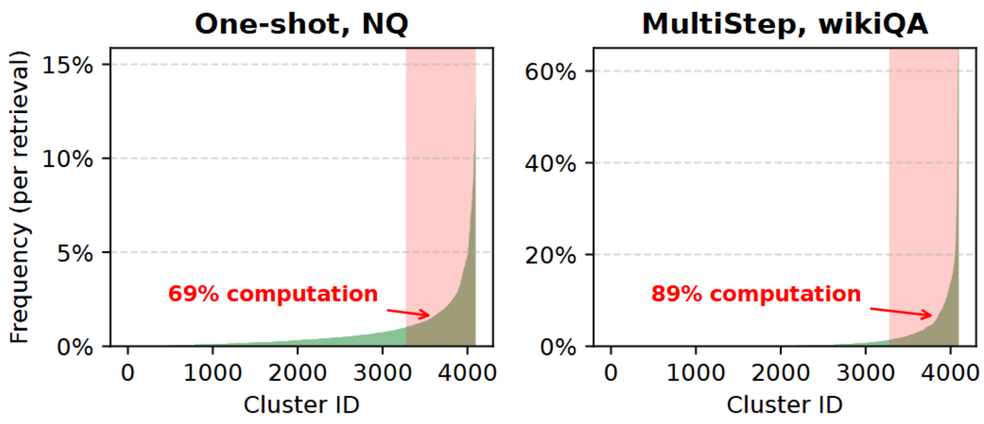
Opportunity – Inter-Request Retrieval Skewness

- **3rd Observation**: skewed cluster access among queries



Various query types

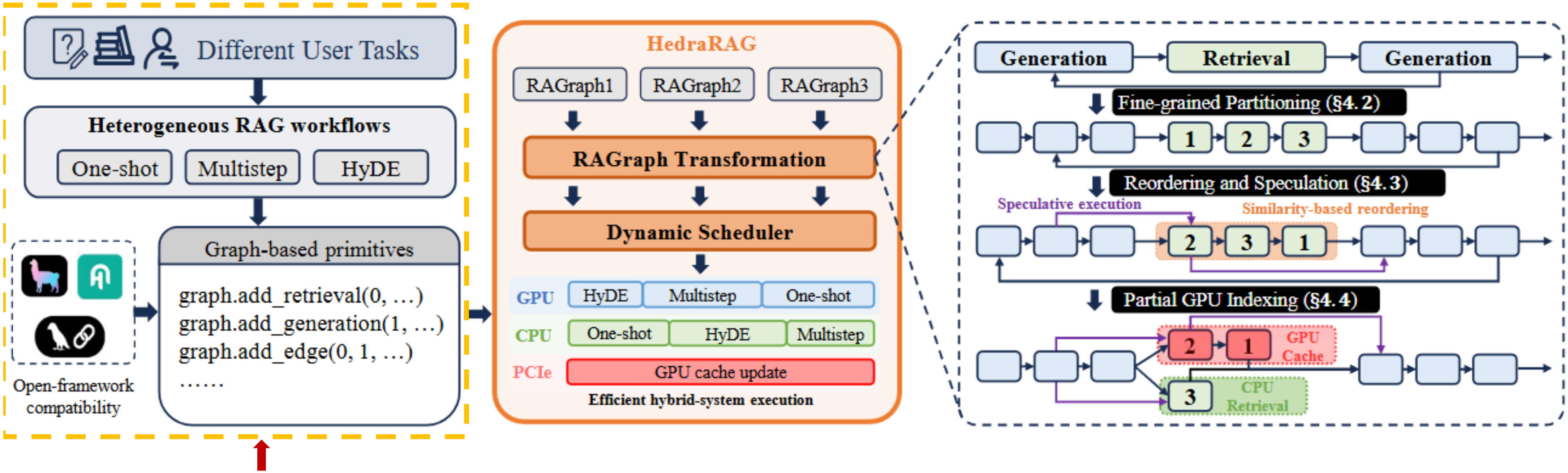
Frequently accessed IVF clusters



A small number of clusters (**20%**) dominate the overall access frequency.

Design – Overview

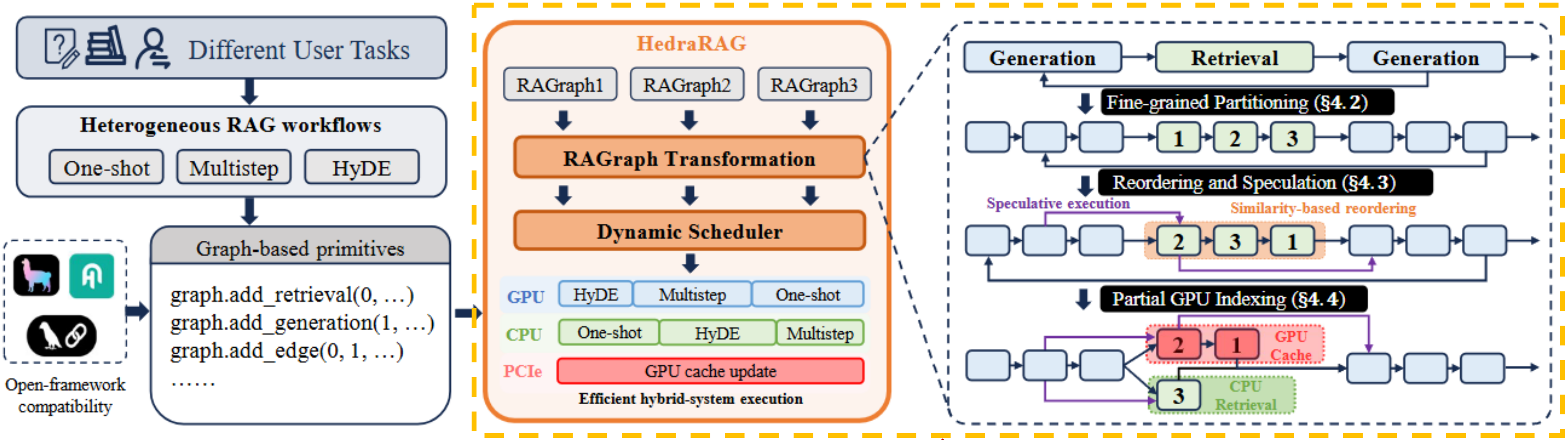
- System Overview
  - **User interface**: Graph-based Workflow Definition



Compatible with Open-sourced frameworks

Design – Overview

- System Overview
  - **User interface**: Graph-based Workflow Definition
  - **Backend server**: Multiple RAG Workflow Co-execution



Throughput & Latency Optimization via Graph-based Transformation

Design – RAGraph

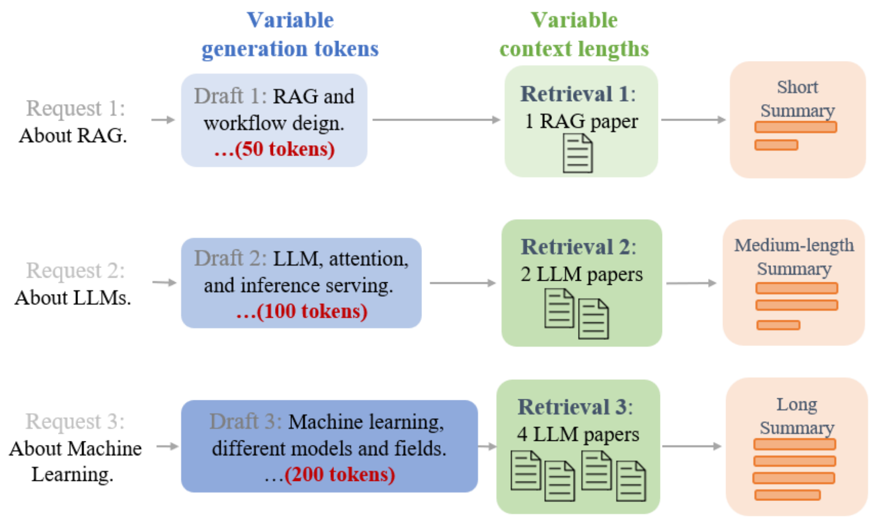
- RAG Specific Abstraction
  - Graph API
    - *add_generation()*
    - *add_retrieval()*
    - *add_edge()*
  - Server API
    - *Server()*
    - *add_request()*

Listing 1. Construct RAG workflows with graph primitives.

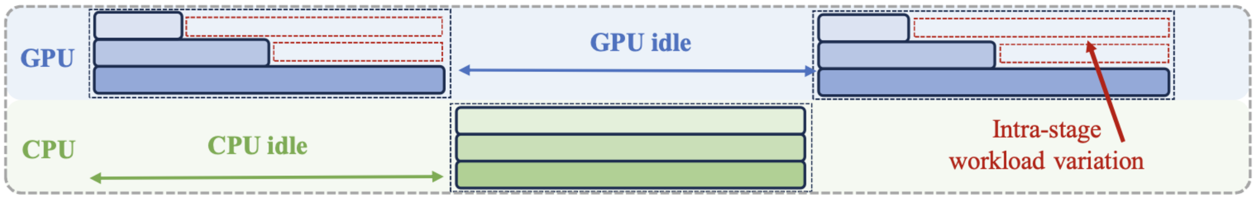
```
from HedraRAG import RAGraph, START, END
from HedraRAG import Server
# HyDE-style workflow
g1 = RAGraph()
g1.add_generation(0, prompt="Generate a hypothesis
                for {input}.", output="hypopara")
g1.add_retrieval(1, topk=5, query="hypopara", output="docs")
g1.add_generation(2, prompt="Answer {query} using {docs}.")
g1.add_edge(START, 0); g1.add_edge(0, 1)
g1.add_edge(1, 2); g1.add_edge(2, END)
# Multistep-style workflow
g2 = RAGraph()
g2.add_generation(0, prompt="Decompose {input} into
                subquestions.", output="subquestion")
g2.add_retrieval(1, topk=2, query="subquestion",
                 output="docs")
g2.add_generation(2, prompt="Answer {subquestion}
                using {docs}.")
g2.add_edge(START, 0); g2.add_edge(0, 1); g2.add_edge(1, 2)
g2.add_edge(2, lambda s: 1 if s.get("subquestion") else END)
# Server initiating and execution
s = Server(generator="Llama3-8B", index="IVF4096")
s.add_request("What is RAG?", g1)
s.add_request("Compare RAG with long-context models.", g2)
```

Design – Fine-grained stage partition

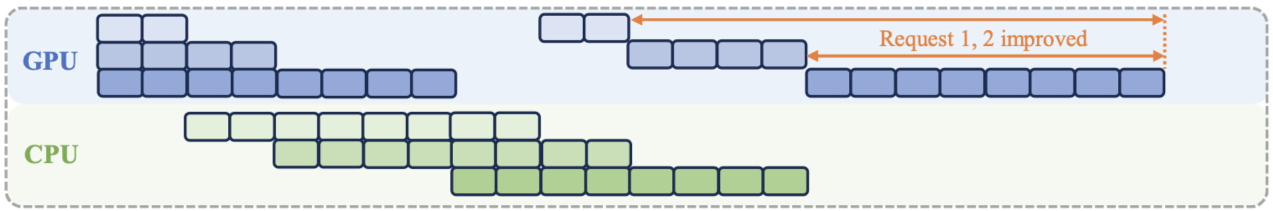
- Partition
  - In generation, each sub-stage comprises several decoding steps.
  - In retrieval, each sub-stage involves searching across one or more clusters



Stage-level Heterogeneity



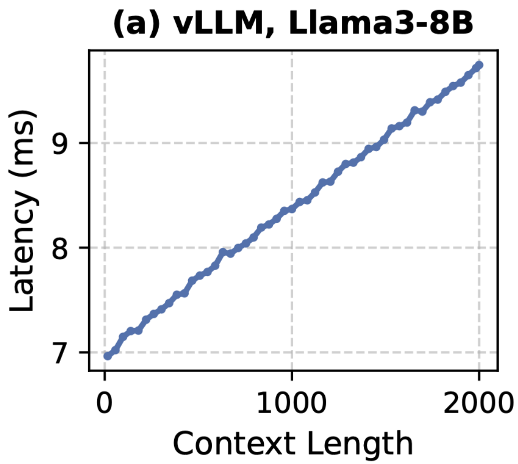
(a) **Modular** and **Sequential** stages (General RAG frameworks)

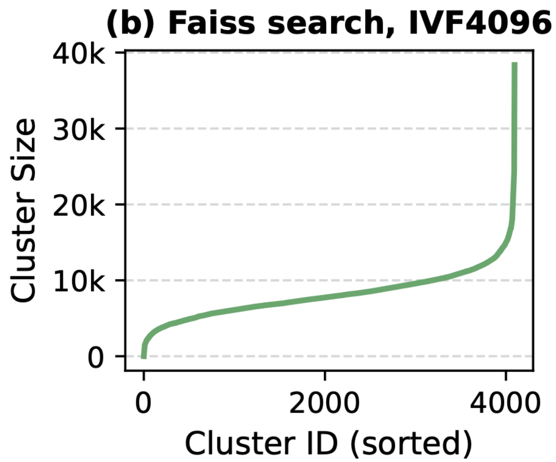


(c) **Fine-grained sub-stage** partitioning and **Dynamic-batched** pipelining (HedraRAG)

Design – Fine-grained stage partition

- Dynamic time-budgeting method based on retrieval requests
  - Before executing a sub-stage, clusters from each retrieval request are incrementally added until a maximum time budget mb is reached.



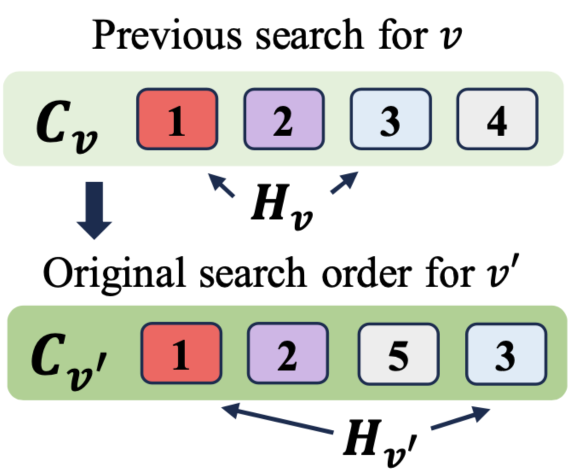


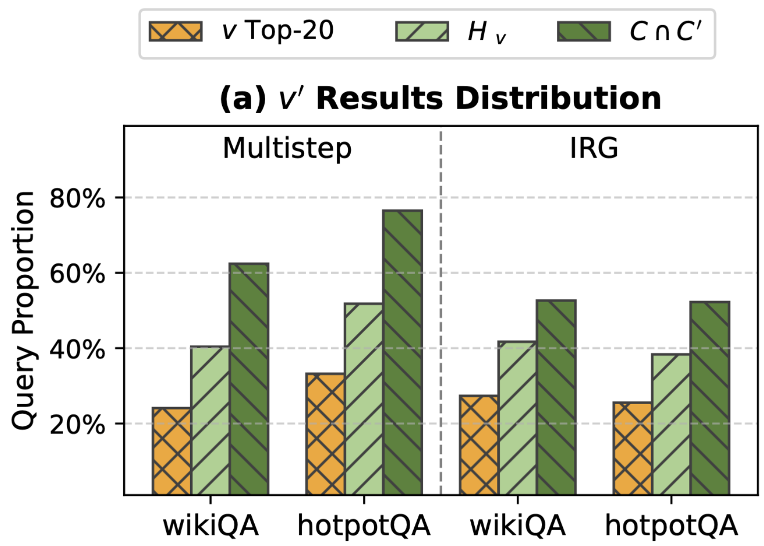
$$mb = \mathrm{argmax}(\Delta_l), \Delta_l = \frac{t_{\mathbf{Retrieval}} - mb}{2} - \frac{t_{\mathbf{Retrieval}}}{mb}\beta$$

- Δl: expected latency improvement
- β: denotes the CPU overhead of request scheduling
- $t_{\mathrm{Retrieval}}$: denotes the average time of retrieval stage

Design – Similarity-Aware Search Optimization

- 3 locality-based observations related to semantic similarity:
  - The search results of v' tend to be included within the search results of v with a larger top-k
  - When the search results of v are in a cluster set Hv, the results of v' also tend to be located in Hv
  - The search results of v' tend to be located in clusters of $C \cap C'$.

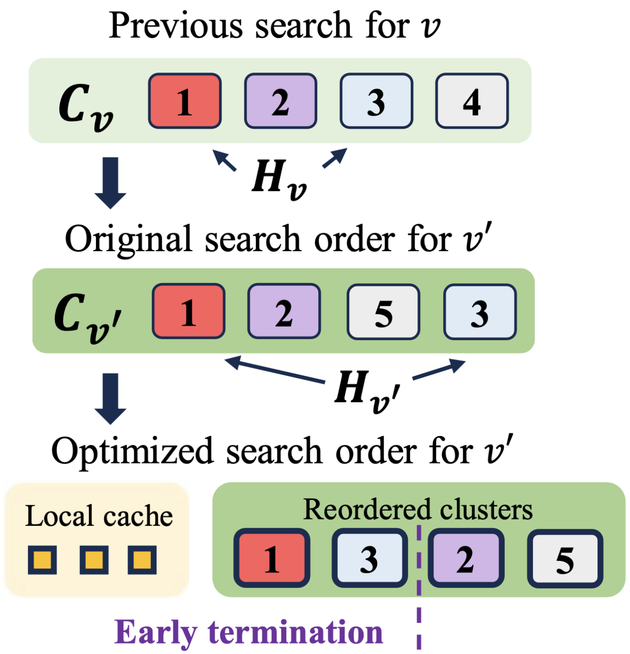


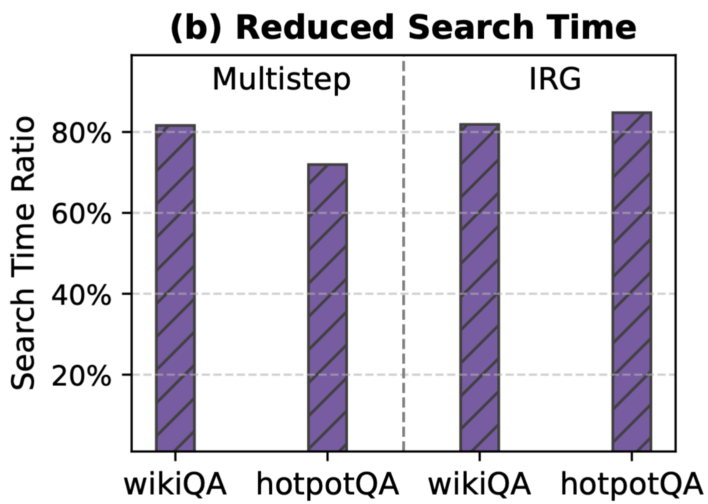


Design – Similarity-Aware Search Optimization

- Cache search information
  - For each retrieval in a request, a set of larger top-k results of v (20 in practice)

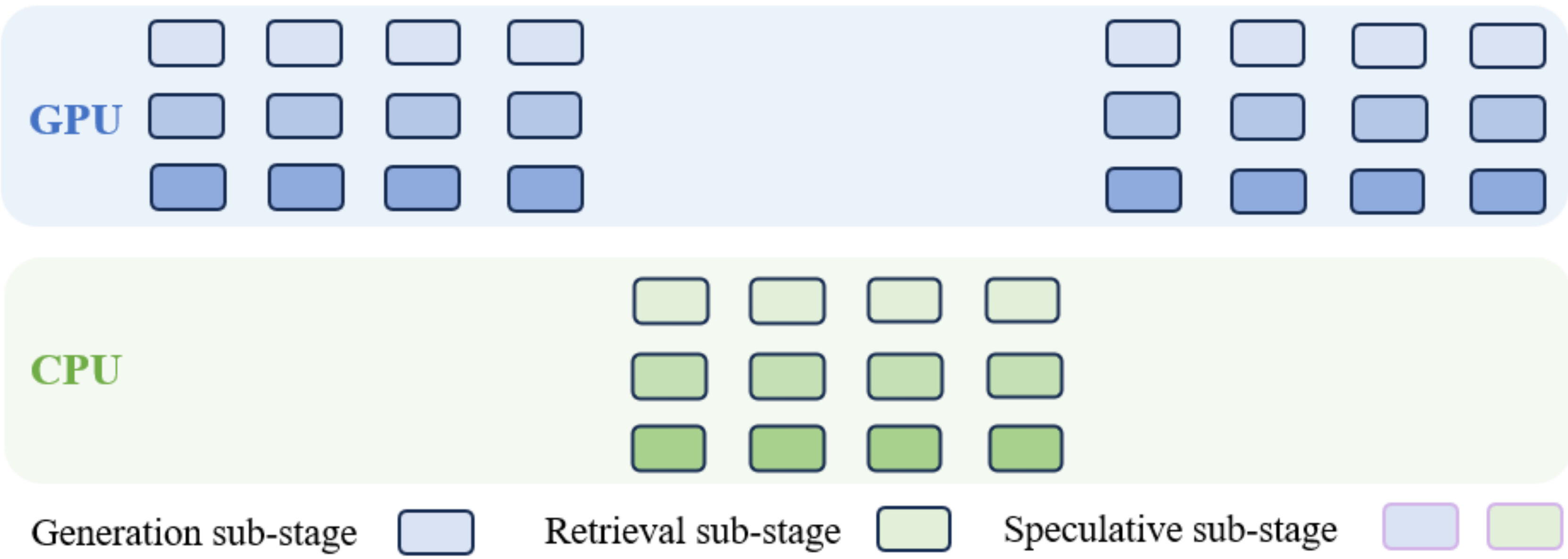
- Reorder of search flow for v'
  - Local cache of v
  - Cluster
    - $Hv \cap C'$
    - $(C - Hv) \cap C'$
    - Remaining clusters





Design – Similarity-Aware Search Optimization

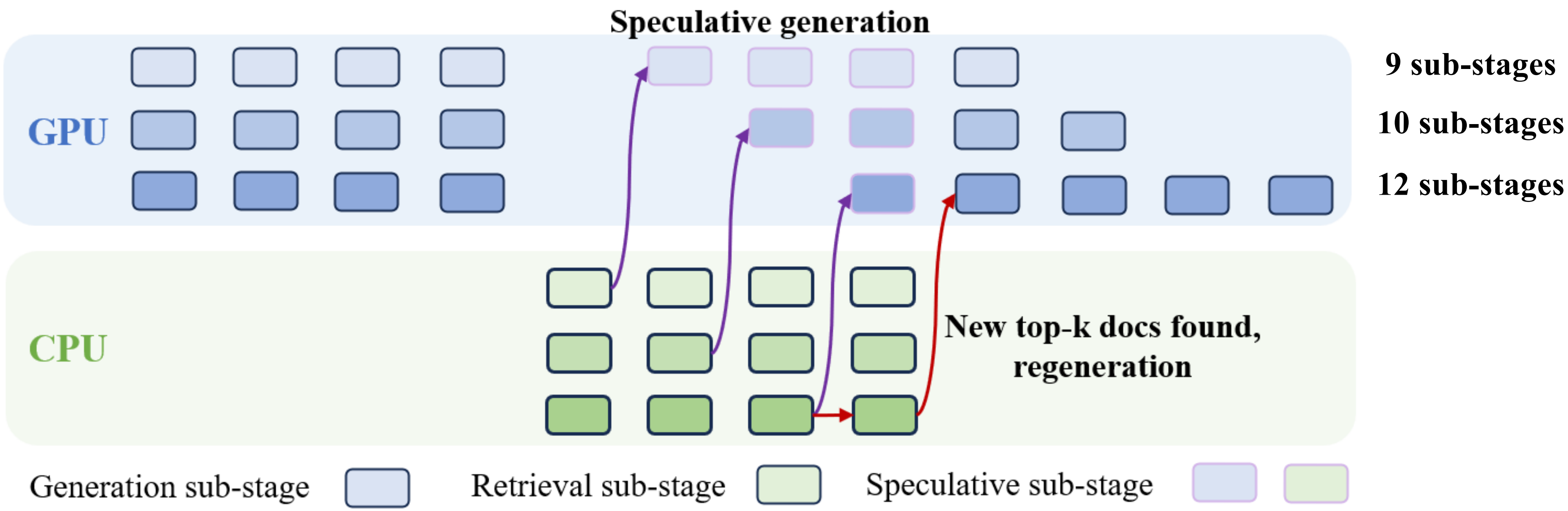
- Speculative Execution
  - **Non-overlapping** execution (Each stage with 4 sub-stages)



12 sub-stages each

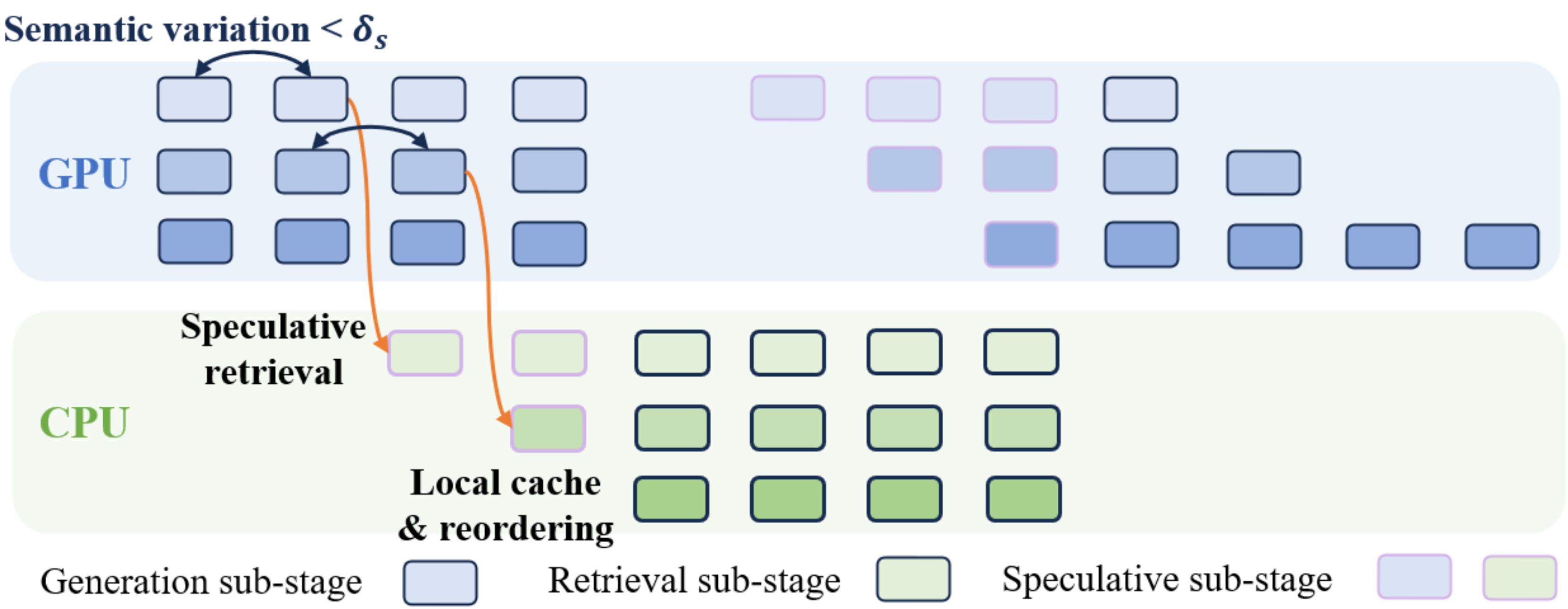
Design – Similarity-Aware Search Optimization

- Speculative Execution
  - Generation with **partial retrieval** (lower search costs)



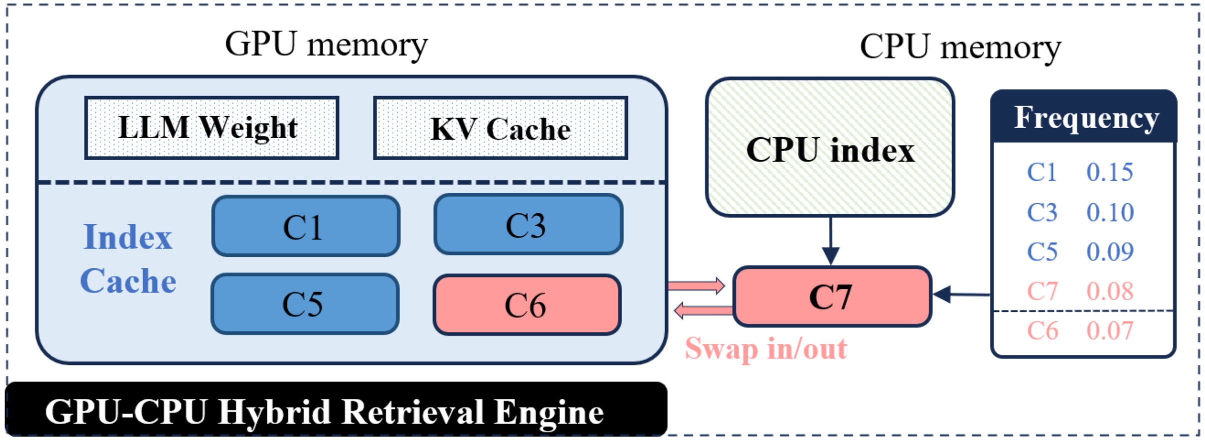
Design – Similarity-Aware Search Optimization

- Speculative Execution
  - Generation with **partial retrieval** (lower search costs)
  - Retrieval with **partial generation** (with caching & cluster reordering)



Design – Partial GPU indexing

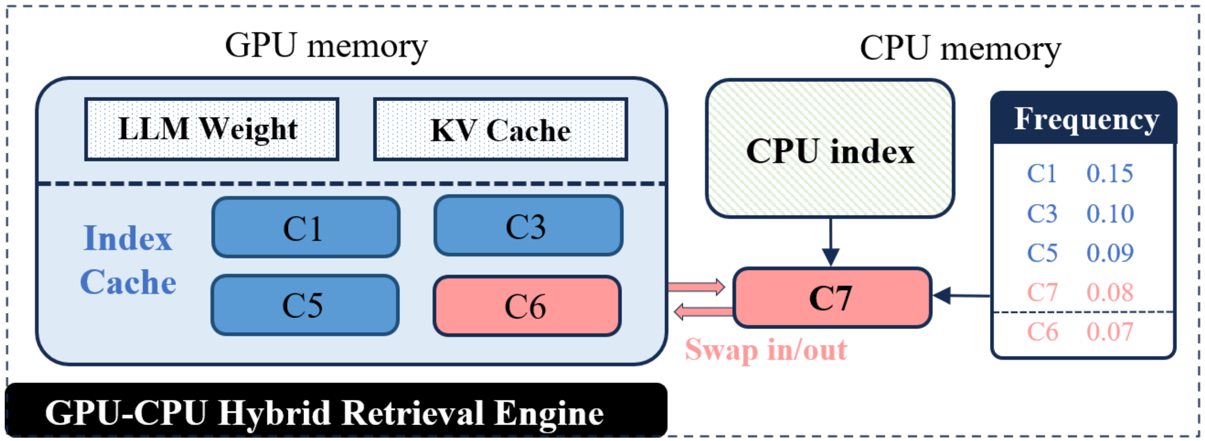
- Dynamic GPU cluster cache



LLM & Index cluster memory coordination

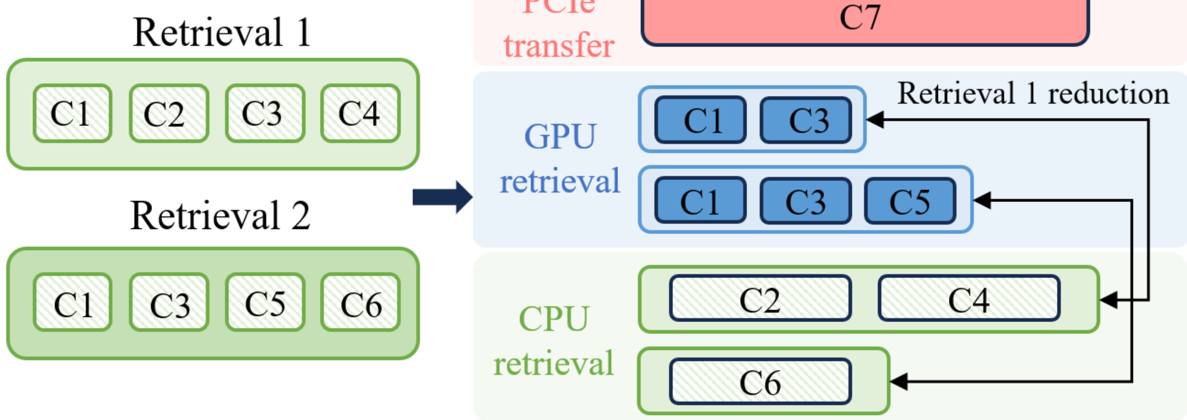
Design – Partial GPU indexing

- Dynamic GPU cluster cache



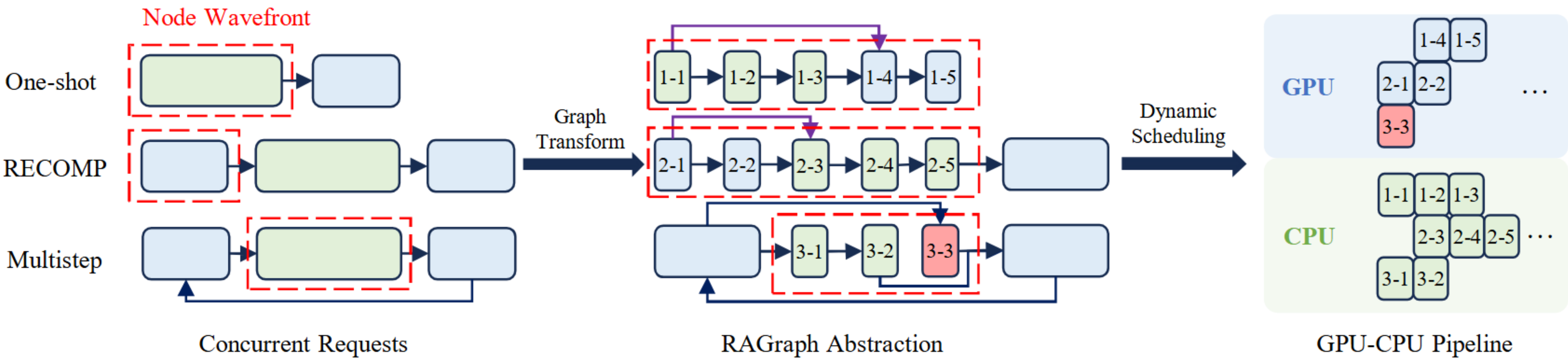
LLM & Index cluster memory coordination

GPU / CPU collaborated vector search



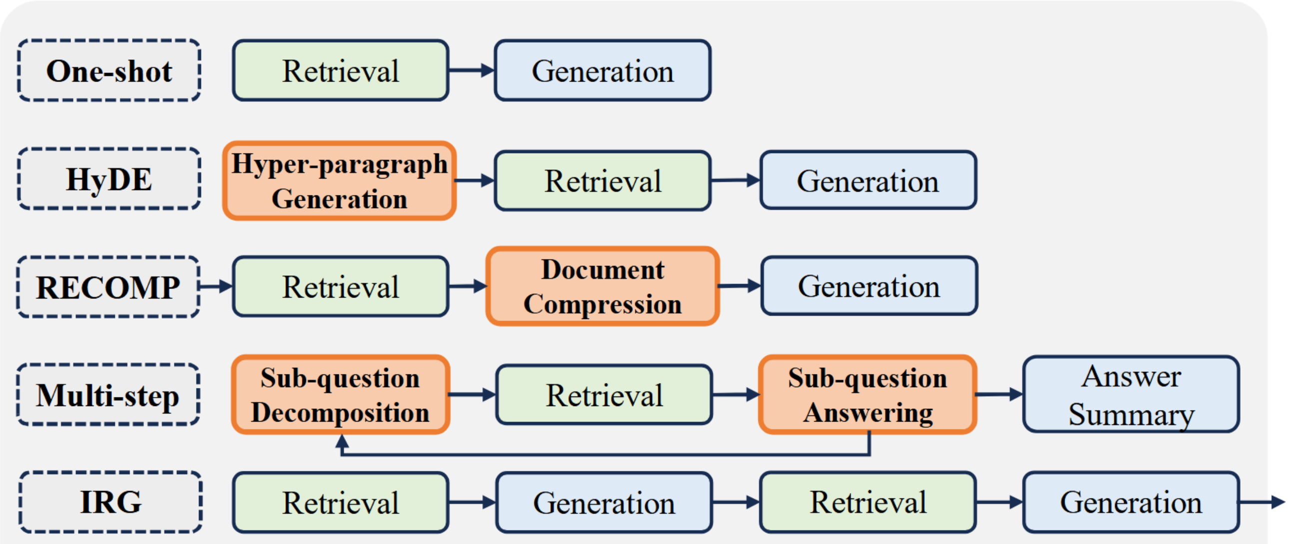
Design – Put them together

- Sub-graph extraction with **Node Wavefronts**
- **Multiple-graph** transforming & scheduling



Evaluation – Setup

- 5 heterogeneous RAG workflows



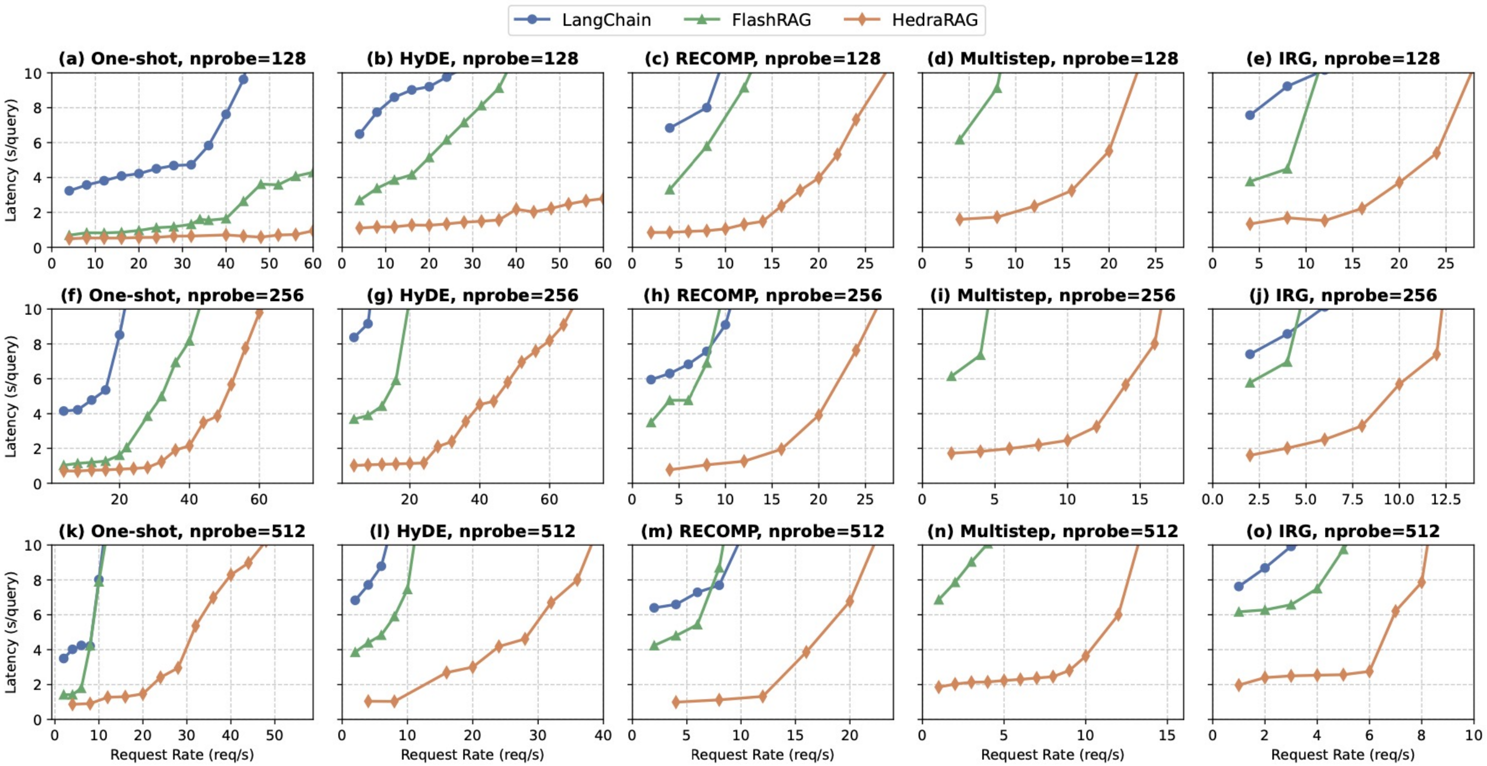
- Baseline: LangChain, FlashRAG (vLLM + Faiss)
- Platform: NVIDIA H100 80GB + AMD EPYC 9534 64-core CPU

Evaluation – Setup

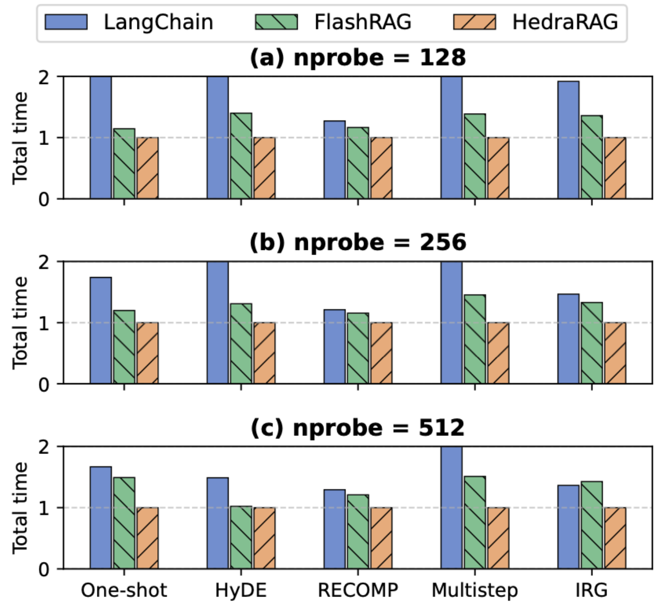
- Model: Llama 3.1 (8B)/ Llama2 (13B)/ OPT (30B)
- Corpus
  - Primary retrieval corpus: Wikipedia passages (38M documents)
  - Embedding mode: e5_large (1024-dimensional)
  - Index: IVF4096
    - Nprobe: 128/256/512
    - Top-k: 1
- Query dataset:
  - NaturalQuestions (NQ)
  - 2WikiMultiHopQA (wikiQA)
  - HotpotQA

Evaluation – Overall

- Online: HedraRAG reduces request latency by 2.2×-18.2×, 3× higher request rates
- Offline: Achieving speedups of 3.5× and 1.3× over LangChain and FlashRAG



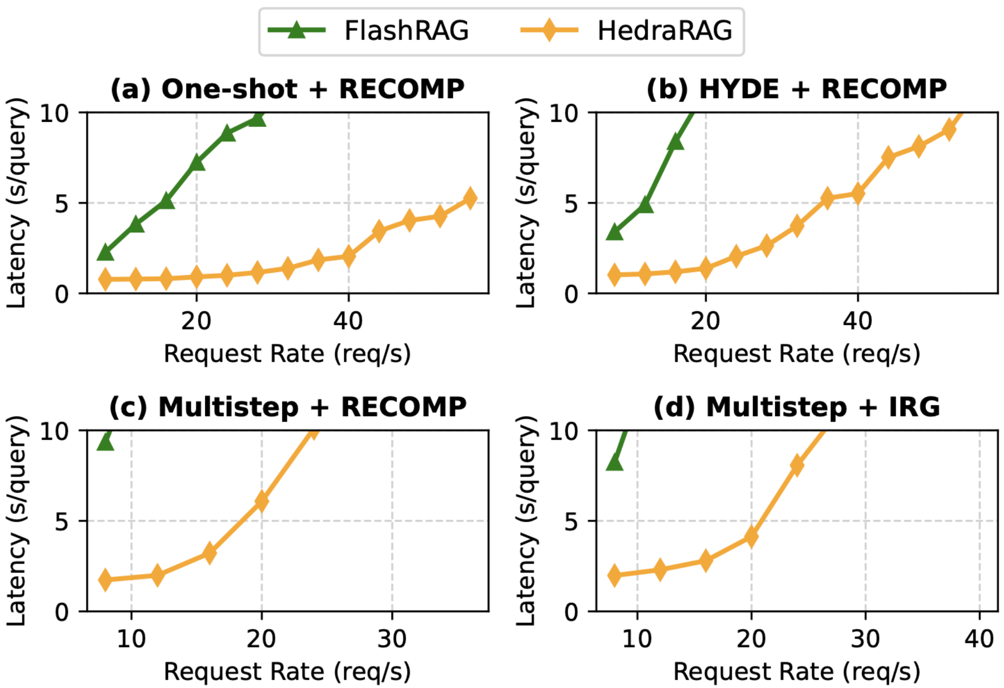
Online



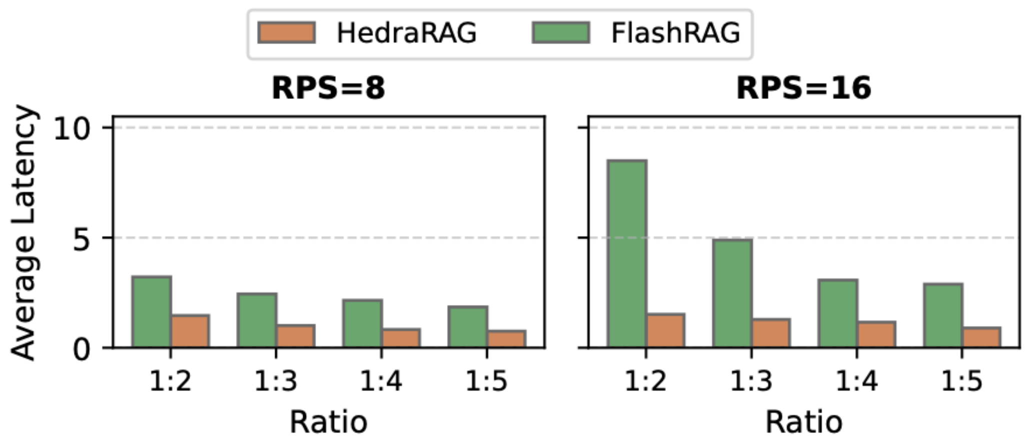
Offline

Evaluation – Overall: hybrid workflows

- >3.3x throughput for multiple workflows
- Achieving up to 5.6× latency reductiotion



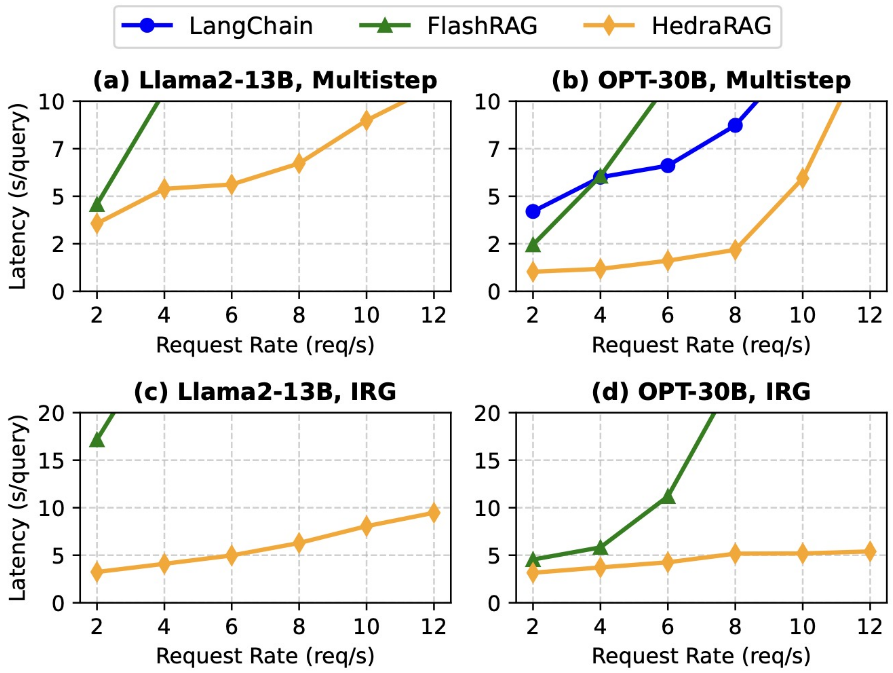
Online Throughput improvement



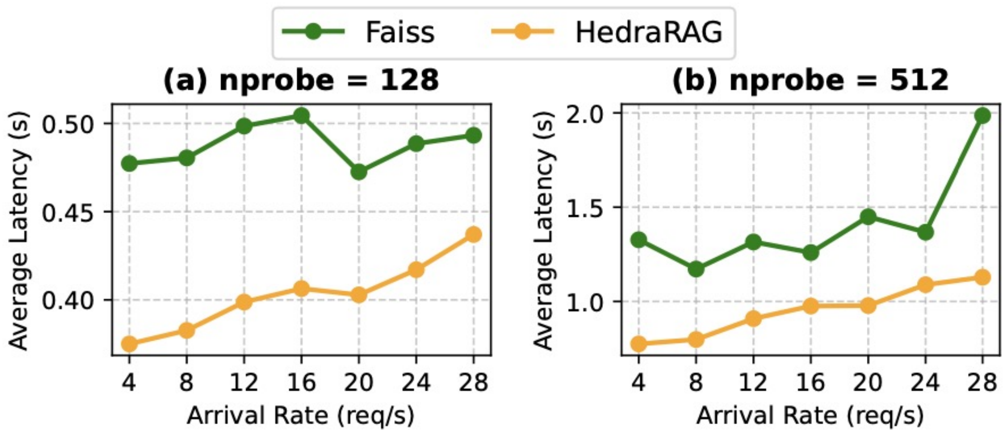
Capability under different
Multistep : One-shot query ratios

Evaluation – Breakdown

- > 1.5x Throughput
- Achieving a reduction of 1.09× to 1.77×



Other LLMs

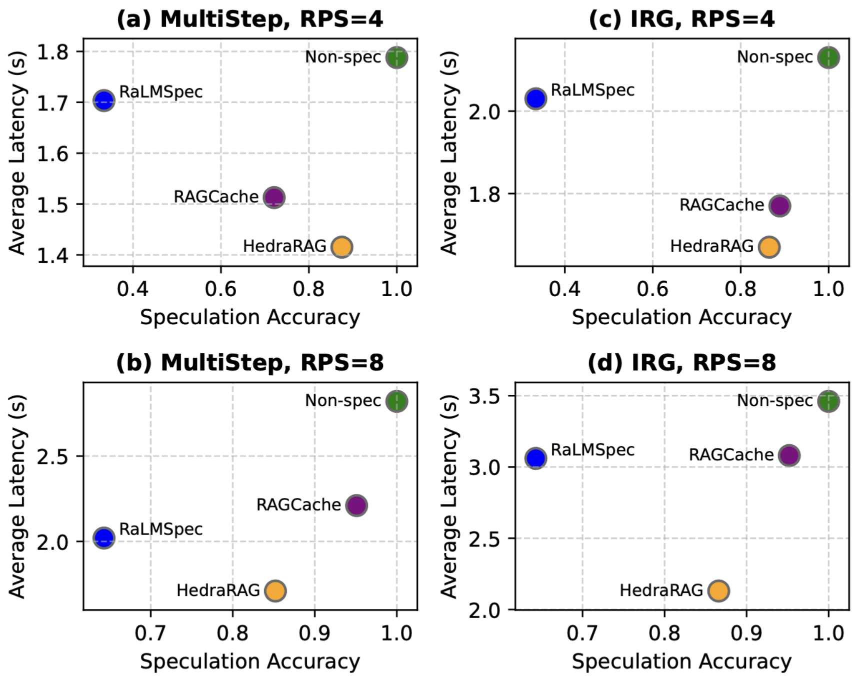


Dynamic partitioning and pipelining

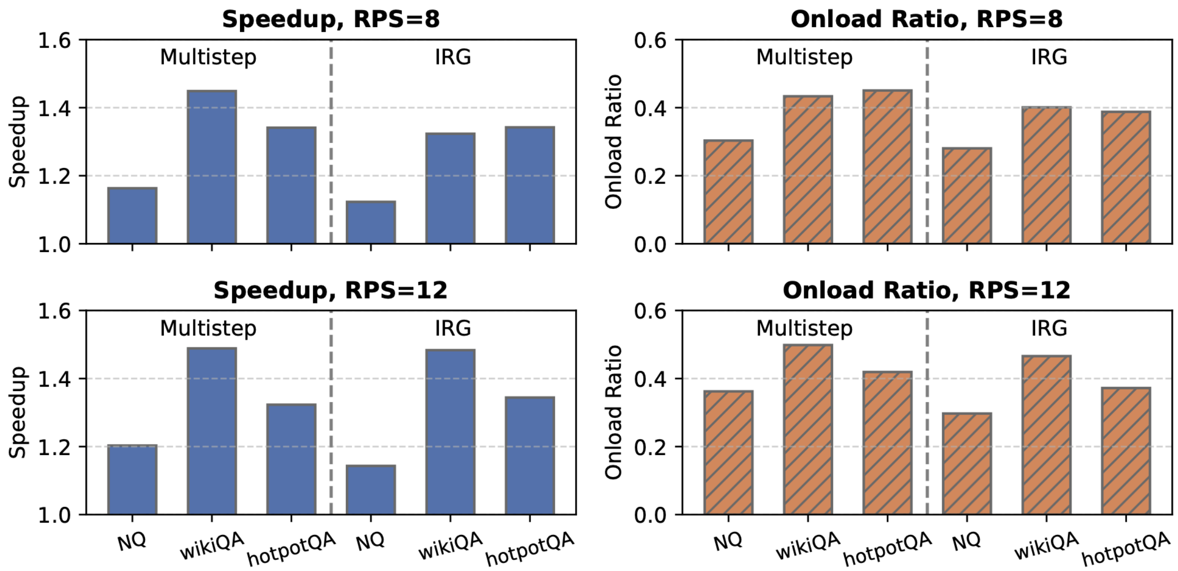
Evaluation – Reordering and Speculation

- Speedup ranging from 1.06× to 1.62×

- Speedup ranging from 1.12× to 1.49×
  - Nprobe: 512



Reordering and Speculation



Partial GPU indexing

Summary

- Our contribution
  - A serving framework to coordinate LLM and vector search
  - Graph-based workflow definition, optimizing and scheduling
  - 3 key techniques to optimize complex & concurrent RAG workflows