Grounding LLM Knowledge

Sreya Gogineni, Justin Paul, Sarah Stec, Brandon Zhang

“Grounding”

S what is the world record for crossing the english channel entirely on foot

The world record for crossing the English Channel entirely on foot is held by Christof Wandratsch of Germany, who completed the crossing in 14 hours and 51 minutes on August 14, 2020. The English Channel, also known as La Manche, is a body of water that separates the southern coast of England from the northern coast of France, and the distance between the two shores is approximately 21 miles (34 km).

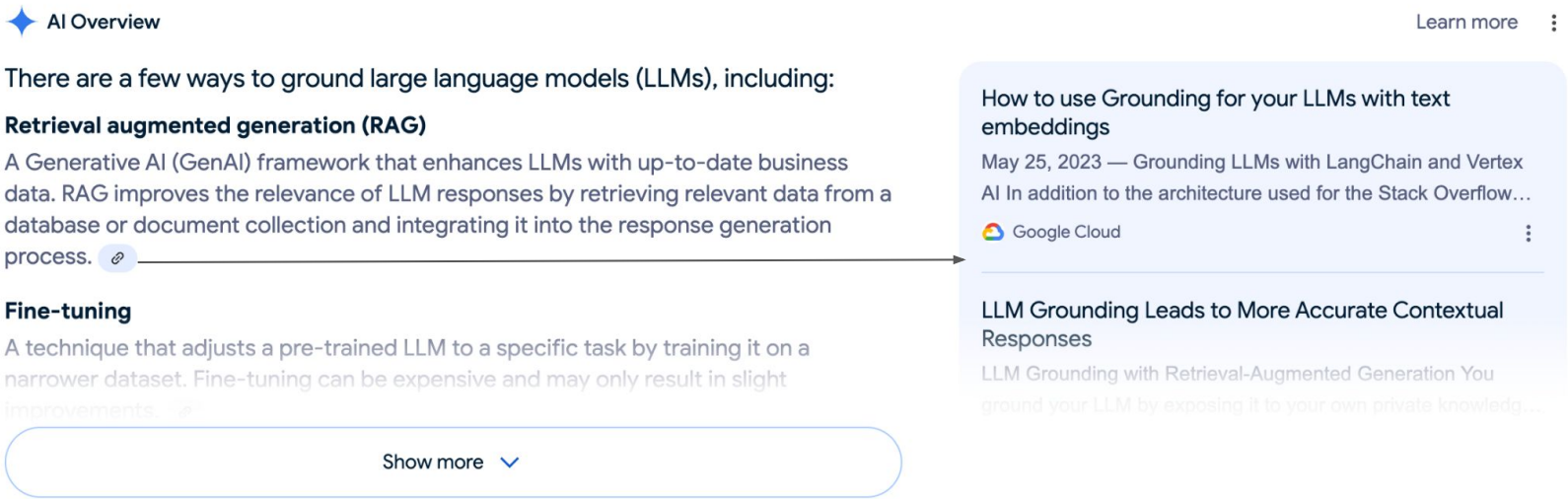
Many people have attempted to cross the channel on foot, but it is a very challenging and dangerous task, due to the strong currents and the chilly water temperature. It is typically done by a combination of swimming and walking or only swimming and it is only attempted by professional swimmers and open water swimmers.

Regenerate response

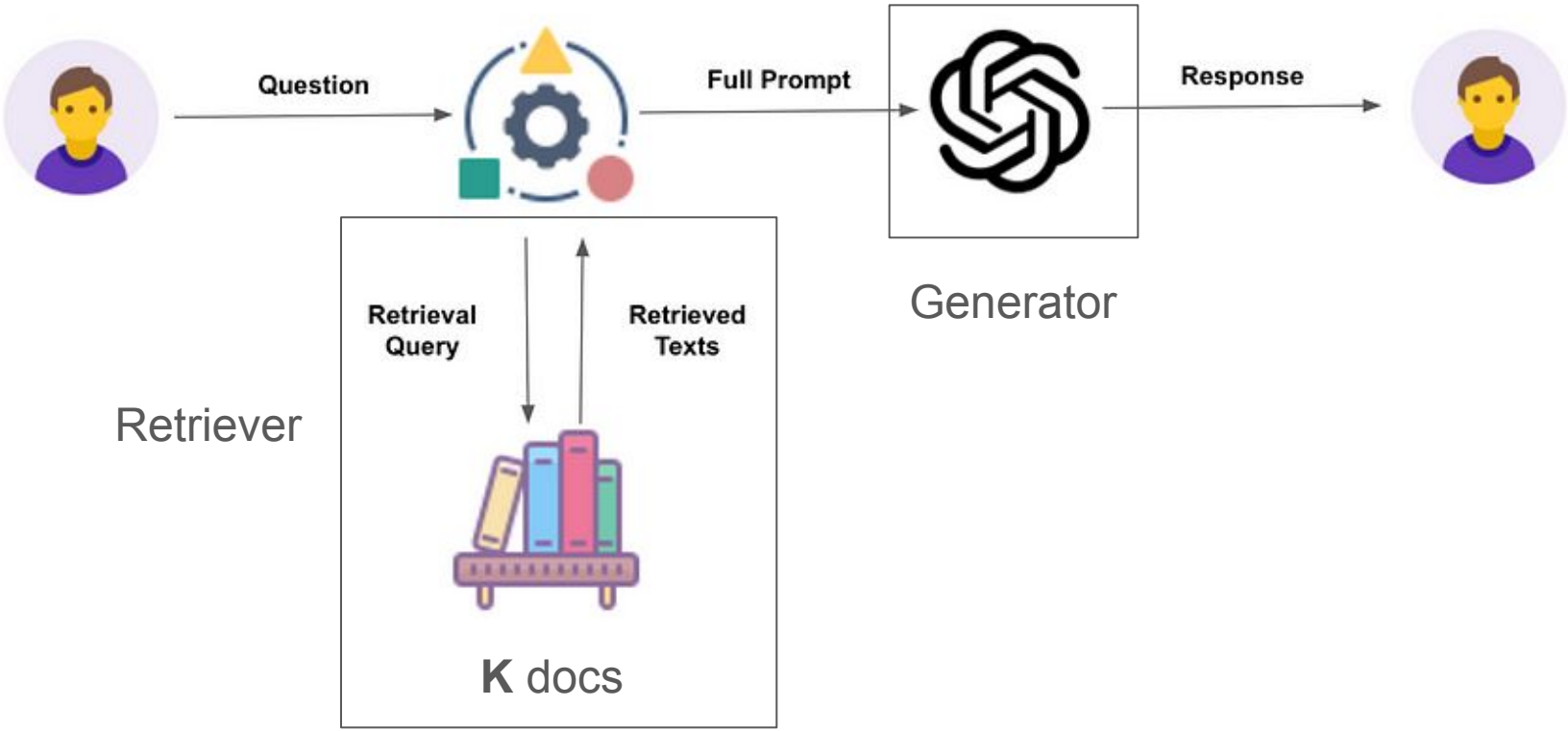
We want the model to produce **relevant** and **accurate** responses

How do we ground responses?

- Fine-tuning
- Chain-of-thought
- Retrieval-Augmented Generation (RAG)
  - **Self-RAG**



Retrieval-Augmented Generation (RAG)

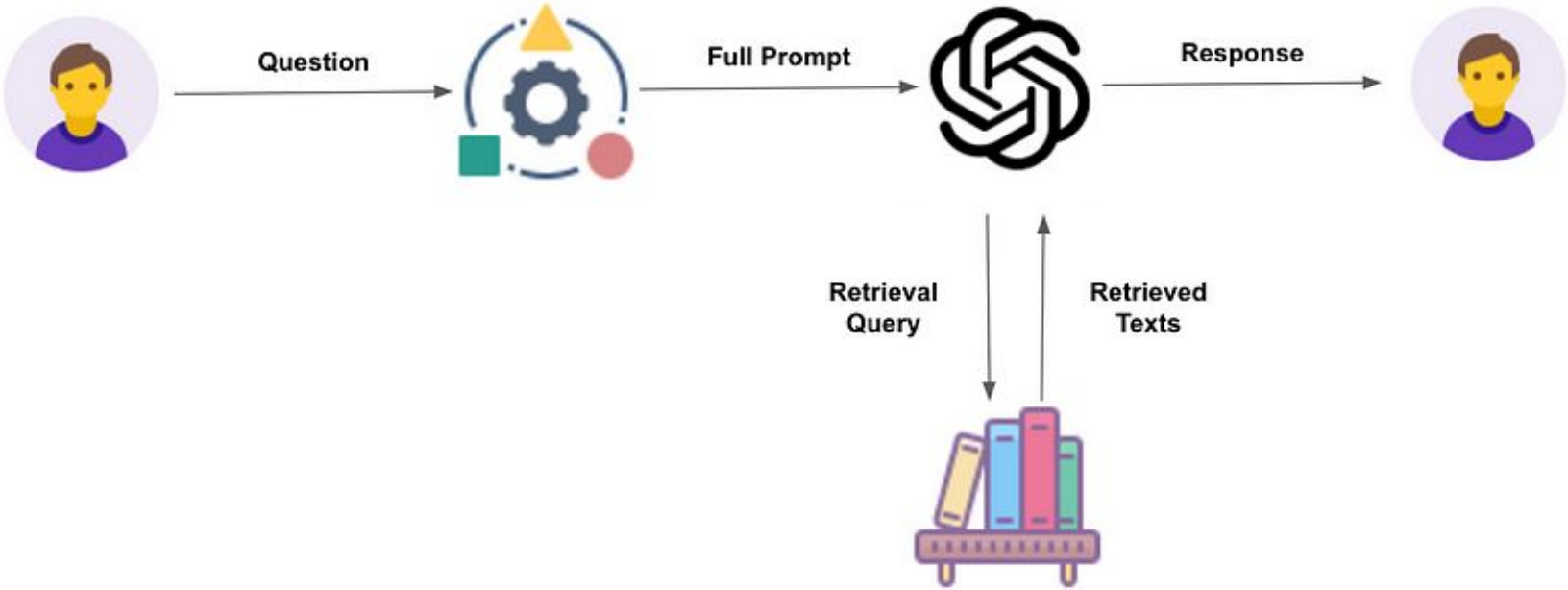


	Finetune	RAG
Specialization	**Highly specialized**	Depends on data sources
Data Sourcing	Unable to cite original source	**Can cite doc info used**
Scalability	Retrain/fine-tune for new data	**Easily update data sources**
Inference	**Feed Directly to Model**	Query relevant docs (latency)

Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, Hannaneh Hajishirzi

Self-Retrieval-Augmented Generation (Self-RAG)



“On-demand retriever”

Self-RAG Improvements

RAG

- Retrieving unnecessary sources adds overhead
- Retrieved sources aren’t always relevant
- Produced output may not be consistent with the source or useful to the prompt

Self-RAG

- Retrieve sources on demand (only when the generator model decides more information is necessary)
- Produce outputs based on different sources in parallel and choose the output that is most
  - relevant to the prompt
  - useful to the prompt
  - consistent to its source

Component Models of Self-RAG

- Generator model: Produce output based on prompt, previous output, and retrieved documents

- Retrieval model: Find sources related to the prompt and previous output

Key idea:
Self Reflection Tokens

Self-reflection tokens

The model is trained to generate **self-reflection** tokens intermittently throughout its normal output.

Self-reflection tokens

Retrieve	Based on *the prompt* and *previous output*, should another source be retrieved? **Yes / No**
IsRelevant	Is the source *relevant* to the prompt? **Yes / No**
IsSupported	Is the output supported by the associated source? **Fully / Partially / Not supported**
IsUseful	Is the output a useful response to x? **1 / 2 / 3 / 4 / 5**

Self-reflection tokens

Retrieve	Based on *the prompt* and *previous output*, should another source be retrieved? **Yes / No**
IsRelevant	Is the source *relevant* to the prompt? **Yes / No**
IsSupported	Is the output supported by the associated source? **Fully / Partially / Not supported**
IsUseful	Is the output a useful response to x? **1 / 2 / 3 / 4 / 5**

Critique tokens

Self-reflection tokens

The model is trained to generate **self-reflection** tokens intermittently throughout its normal output.

- Produce a `Retrieve` token before producing a segment of text

Self-reflection tokens

The model is trained to generate **self-reflection** tokens intermittently throughout its normal output.

- Produce a `Retrieve` token before producing a segment of text
- If a source was retrieved, produce the `IsRelevant`, `IsSupported`, and `IsUseful` tokens at the end of the segment

Self-reflection tokens

The model is trained to generate **self-reflection** tokens intermittently throughout its normal output.

- Produce a `Retrieve` token before producing a segment of text
- If a source was retrieved, produce the `IsRelevant`, `IsSupported`, and `IsUseful` tokens at the end of the segment
- If a source is not retrieved, only produce the `IsUseful` token

Generating Self-RAG Training Data

- The Self-RAG generator model trained on a large corpus of text interspersed with self-reflection tokens

Input Synthesize a poem based on this prompt:"The sun is setting" (from `stanford_alpaca`)
Output [Retrieve=No]The stars begin to twinkle bright the trees, scattered throughout the night; [Retrieve=No]The stars begin to twinkle bright the trees, scattered throughout the night [Retrieve=No] They flicker like a silent dance, A peaceful moment in its trance. [Retrieve=No] They flicker like a silent dance, A peaceful moment in its trance. [Retrieve=No] A chorus of crickets sing a lullaby, To every creature that roams the sky [ISUSE =5]

- The original text was annotated with self-reflection tokens by a *critic* model

Training the Critic Model

- Extremely large proprietary language models (like GPT-4) are very good (comparable to human-eval level) at predicting self-reflection token values

  GPT-4 retrieve token prediction prompt: "***Given an instruction, make a judgment on whether finding some external documents from the web helps to generate a better response***."

- Relying on a large proprietary model for annotating all the data is **costly**

- A *smaller critic model* can be trained an a small corpus of text annotated with GPT-4's self-reflection token predictions

Step 1: Retrieve On Demand

- Predict a YES or NO for the **retrieve token** for whether a source would be helpful to answer the prompt

Prompt How did US states get their names?

vs.

Sources describing name origins would be useful

Prompt: Write an essay of your best summer vacation

No sources are necessary for a made up story!

Step 1: Retrieve On Demand

- Predict a YES or NO for the **retrieve token** for whether a source would be helpful to answer the prompt

Step 2: Generate Output

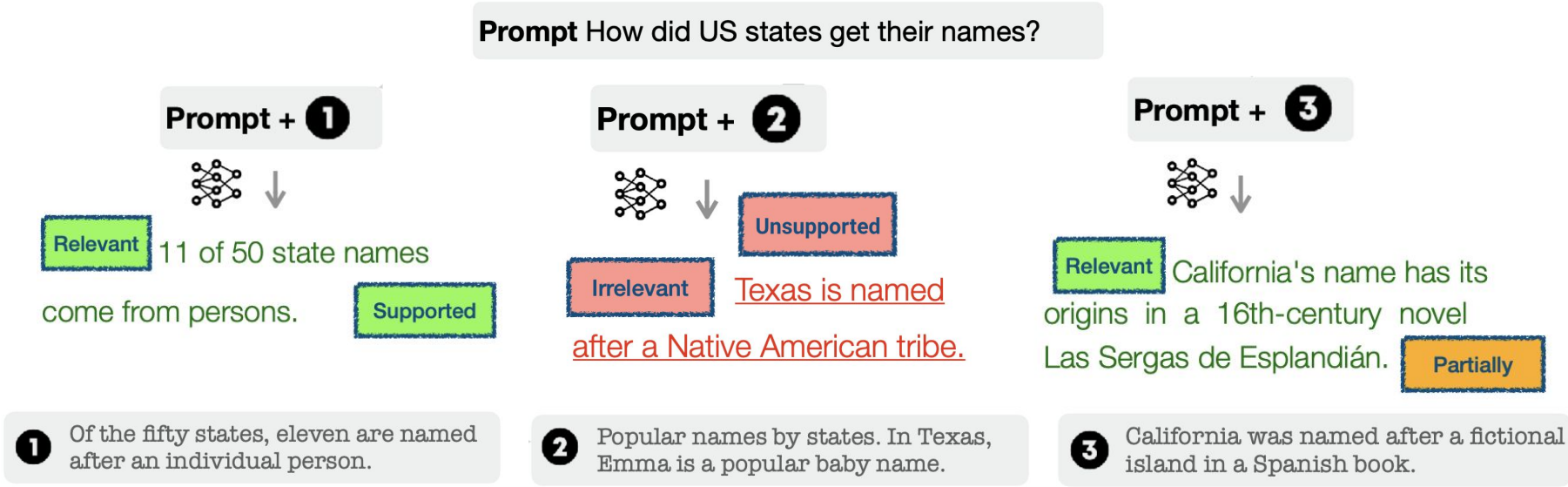
- For each source, produce an output in parallel

Step 3: Critique Sources and Output

- Predict a YES or a NO for the **isRelevant token** for each source in regards to the prompt

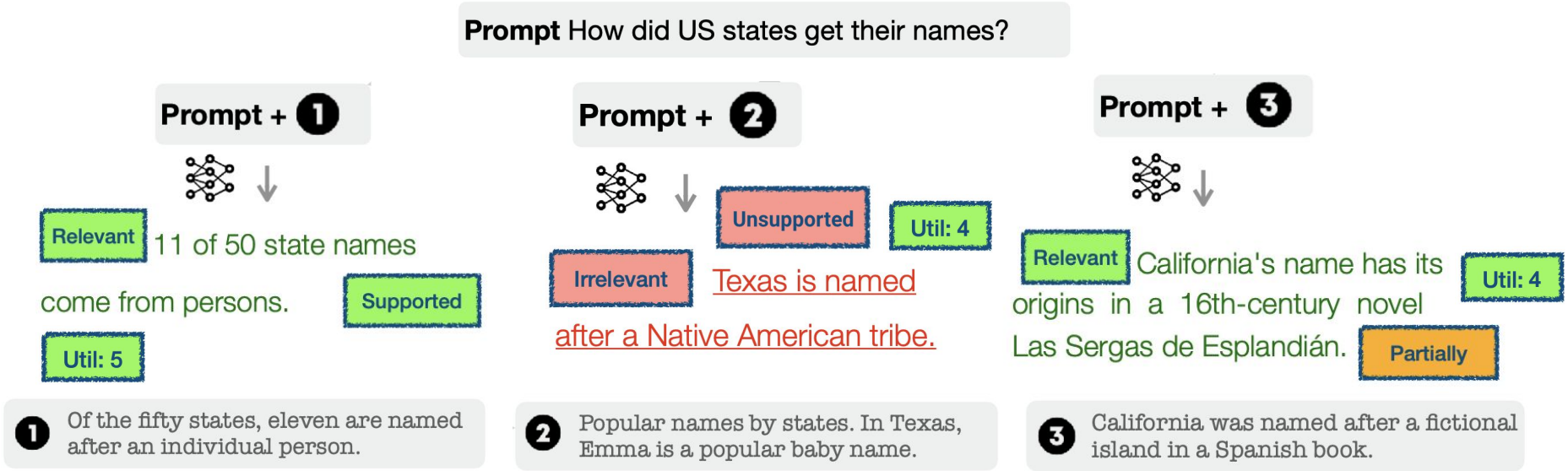
Step 3: Critique Sources and Output

- Predict as **isSupported** token for each output; an output is evaluated as either *supported, partially supported,* or *not supported* by the associated source



Step 3: Critique Sources and Output

- Predict an **isUseful token** for each output, a rating from *1-5* (Util: 5 means the output is highly useful for answering the prompt)



Step 4: Select an output

Rank the outputs based on quality, taking the self reflection into account:

Consider a “critique score” for each output based on the critique token values:

Sum the probabilities of generating the *most desirable value* for each critique token.

(example) **isRelevant token** probability distribution:

The higher the critique score, the more favorable the self reflection of the output.

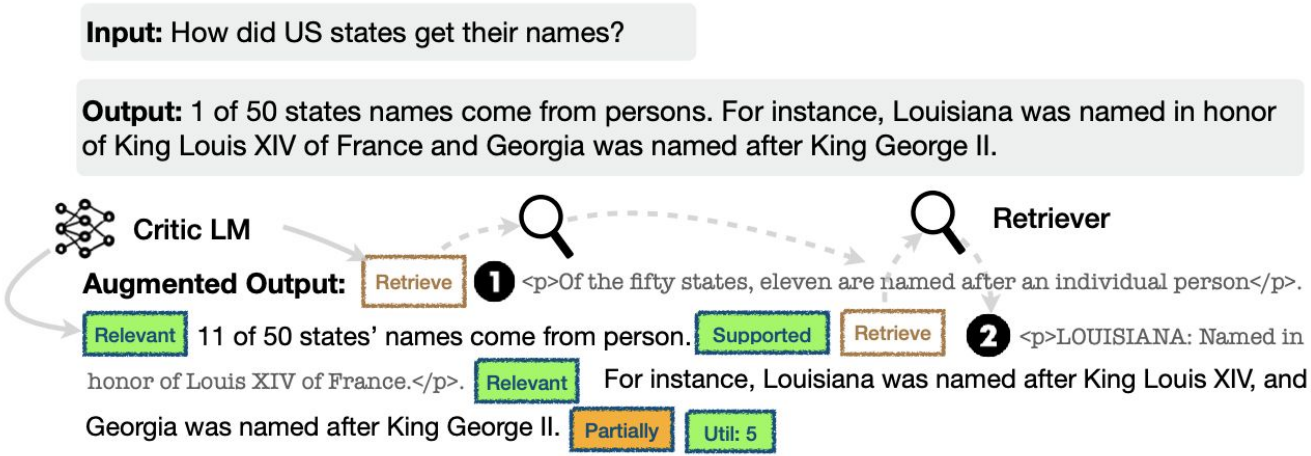
Step 4: Select an output

- Return the most highly ranked output

US states got their names from a variety of sources. 11 of 50 states names are come from persons. 1

Step 5: Repeat the process until the response is complete

- Now, using the prompt and the previously generated segment as input, repeat the process of RAG



Self-RAG Outperforms Baselines With RAG

Paper's self-RAG model augmented Llama-2

Paper's models outperform unaugmented and RAG Llama-2 (among others) on

- factuality
- correctness
- fluency
- citation precision
- citation recall

LM	Short-form PopQA (acc)	Short-form TQA (acc)	Closed-set Pub (acc)	Closed-set ARC (acc)	Long-form generations (with citations) Bio (FS)	ASQA (em)	ASQA (rg)	ASQA (mau)	ASQA (pre)	ASQA (rec)
Baselines without retrieval										
Llama2$_{7B}$	14.7	30.5	34.2	21.8	44.5	7.9	15.3	19.0	–	–
Alpaca$_{7B}$	23.6	54.5	49.8	45.0	45.8	18.8	29.4	61.7	–	–
Llama2$_{13B}$	14.7	38.5	29.4	29.4	53.4	7.2	12.4	16.0	–	–
Alpaca$_{13B}$	24.4	61.3	55.5	54.9	50.2	22.9	32.0	70.6	–	–
CoVE$_{65B}$ *	–	–	–	–	71.2	–	–	–	–	–
Baselines with retrieval										
Toolformer*$_{6B}$	–	48.8	–	–	–	–	–	–	–	–
Llama2$_{7B}$	38.2	42.5	30.0	48.0	78.0	15.2	22.1	32.0	2.9	4.0
Alpaca$_{7B}$	46.7	64.1	40.2	48.0	76.6	30.9	33.3	57.9	5.5	7.2
Llama2-FT$_{7B}$	48.7	57.3	64.3	65.8	78.2	31.0	35.8	51.2	5.0	7.5
SAIL*$_{7B}$	–	–	69.2	48.4	–	–	–	–	–	–
Llama2$_{13B}$	45.7	47.0	30.2	26.0	77.5	16.3	20.5	24.7	2.3	3.6
Alpaca$_{13B}$	46.1	66.9	51.1	57.6	77.7	**34.8**	36.7	56.6	2.0	3.8
Our SELF-RAG $_{7B}$	54.9	66.4	72.4	67.3	**81.2**	30.0	35.7	**74.3**	66.9	67.8
Our SELF-RAG $_{13B}$	**55.8**	**69.3**	**74.5**	**73.1**	80.2	31.7	**37.0**	71.6	**70.3**	**71.3**

Self-RAG Overview

Background: Grounding improves the relevance and reliability of LLM responses

Problem:

- Current solution RAG incurs source retrieval cost even with sources are not necessary
- No reflection on the relevance of sources or quality of output from different sources

Idea: Self-reflection tokens

Solution:

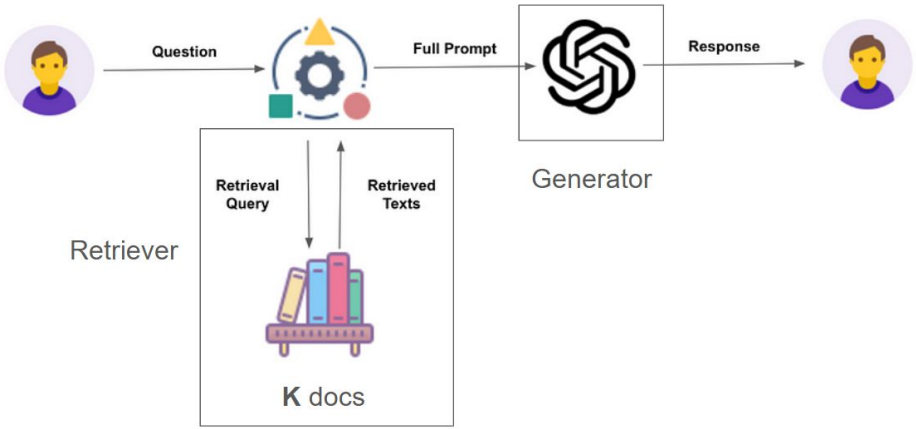
- Allow model to decide if source retrieval is necessary or not
- Improve source relevance, output usefulness and consistency with sources

Fast Vector Query Processing for Large Datasets Beyond GPU Memory with Reordered Pipelining

Zili Zhang, Fangyue Liu, Gang Huang, Xuanzhe Liu, Xin Jin

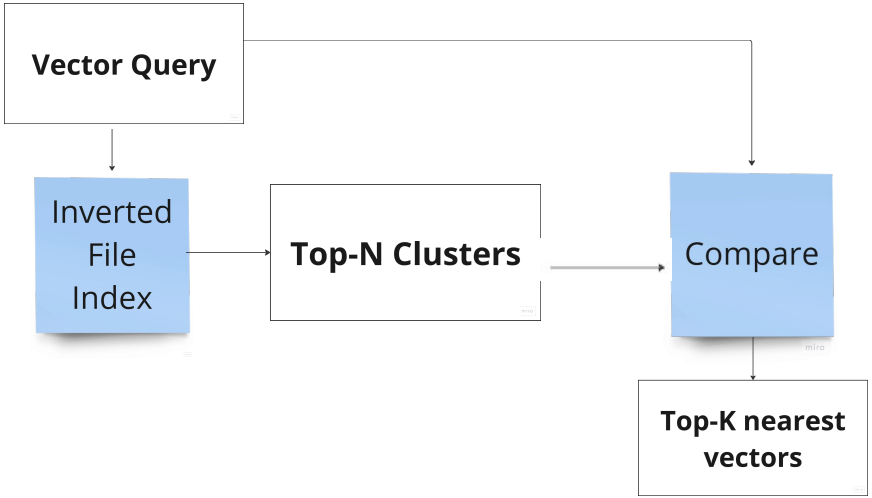
Background - Vector Query

- Unstructured data (images, videos, etc) embedded as high-dimensional vectors
- Vector query finds which documents are most relevant to the query (closest to the query in high-dimensional space)



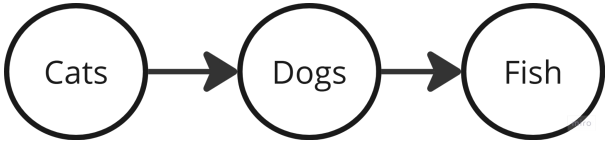
Problem - High query latency on large datasets

- Divide dataset into clusters of related vectors + store in inverted file index (IVF)
- Look up top-N clusters related to query via IVF
- Compare query only against vectors in top-N related clusters



Query: What are common pets?

Top-N clusters of related vectors:



Top-K related vectors:

- Wikipedia article on popular household fish
- News report on popular dog breeds

Key Problem: Limited GPU Memory

Large datasets do not fit in GPU memory

- 750GB TEXT1B dataset vs 80GB A100 memory

Workarounds:

- Run on CPU
  - High latency
- Use enough GPUs to store dataset in GPU memory
  - Due to cluster-based search, low utilization of GPU resources
- Alternate between transmitting and computing on GPU
  - Sequential transmission + computation = low GPU utilization
- CUDA unified memory
  - Vector unaware - GPU page faults for data transmission

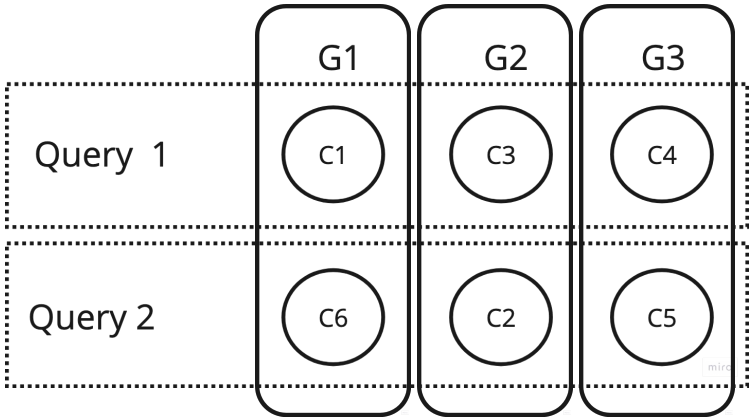
Key idea:
Pipeline GPU transmission and computation

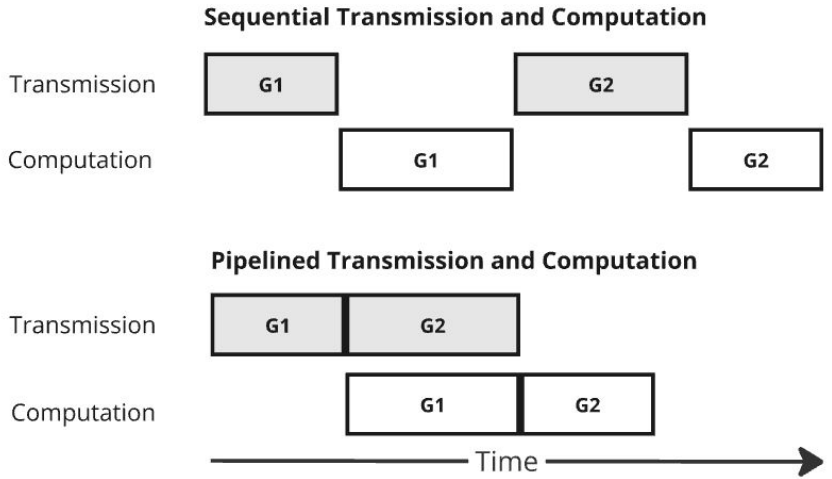
Ci = cluster i
Gj = group j

Pipelining

Overlap transmitting clusters to GPU and processing them to minimize latency

- Process multiple queries together by grouping clusters from different queries (i.e. G1={C1, C6})





Pipelining Challenges

1. Unnecessarily retransmitting clusters

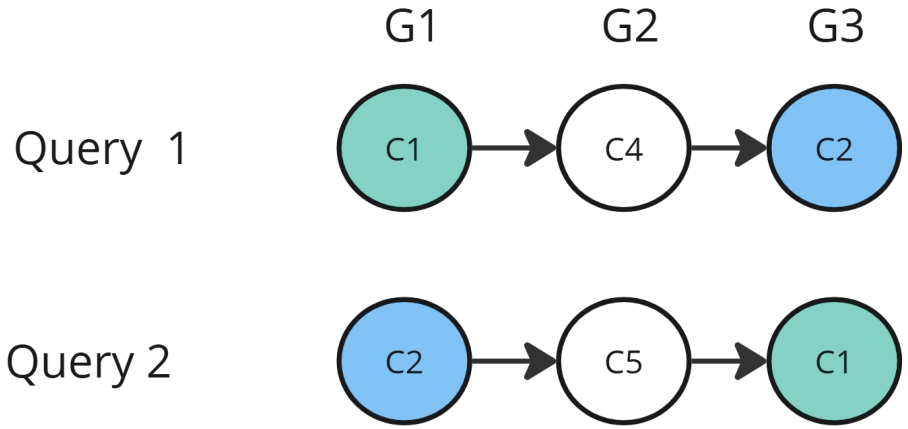
2. Underutilizing GPU compute

3. Difficulty in maximizing overlap between transmitting and processing clusters

Ci = cluster i
Gi = group j

Challenge 1: Cross-query redundant transmission

1. G1={C1, C2} copied to GPU memory
2. Upon transmitting G2={C4,C5}, G1 must be evicted to create space
3. Retransmit G3={C2,C1}, G2 must be evicted



Transmissions for batch of 2 queries

C1 and C2 were transmitted twice!

Ci = cluster i

Challenge 2: Spatial and temporal GPU underutilization

Streaming Multiprocessors (SM) run thread blocks on GPUs and are idle…

- When computation on small clusters finish early (temporal)
- When there are not enough clusters to utilize every SM (spatial)

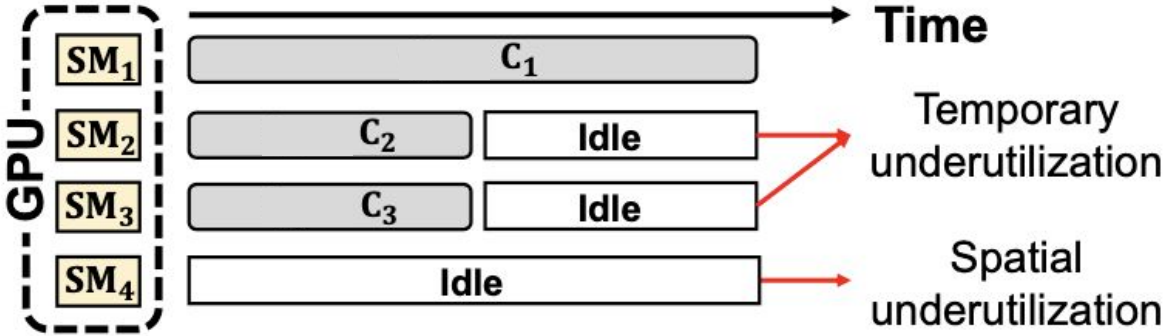


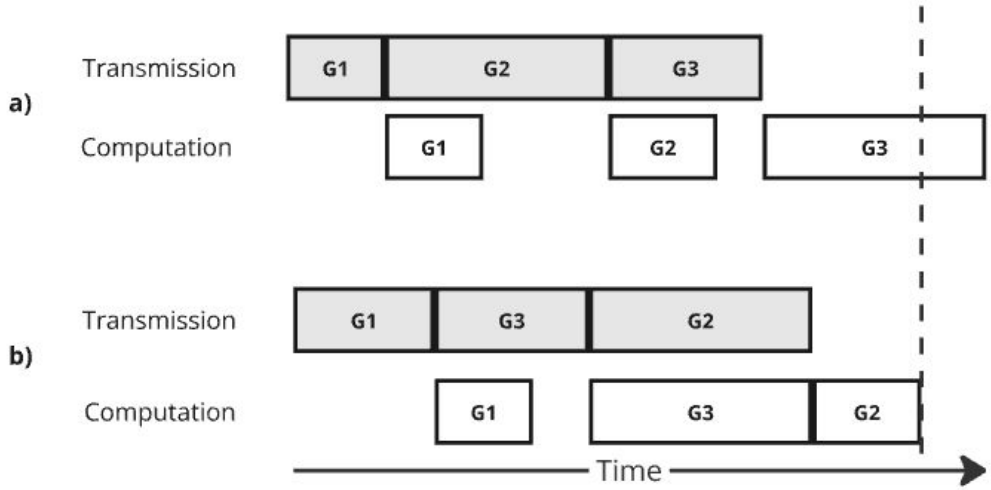
Figure 3: Spatial and temporal GPU underutilization.

Gi = group i

Challenge 3: Maximizing the overlap between computation and transmission

Maximizing overlapping minimizes total latency

Maximizing overlapping requires finding optimal ordering of transmission and computation



Rummy Overview

Pipeline transmission and computation via reordered pipelining technique

Key Strategies:

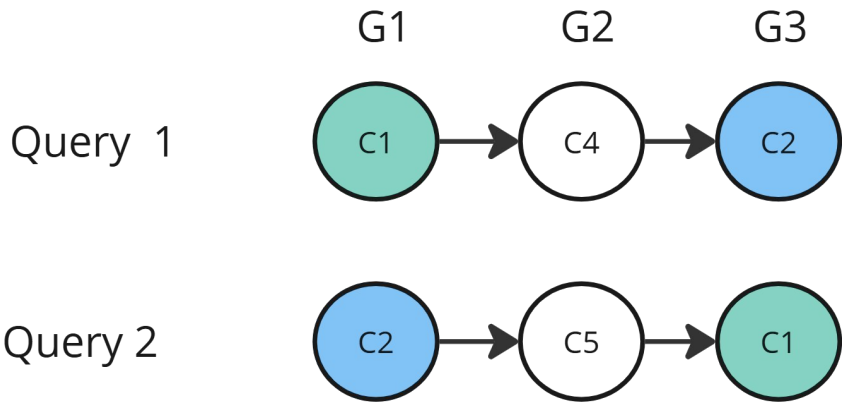
1. Reuse clusters to eliminate redundant transmission
2. Equalize cluster size + split clusters across thread blocks to maximize GPU utilization
3. Reorder groups in pipeline to maximize transmission + computation overlapping

Ci = cluster i
Gi = group j

Query Plan Retrofitting

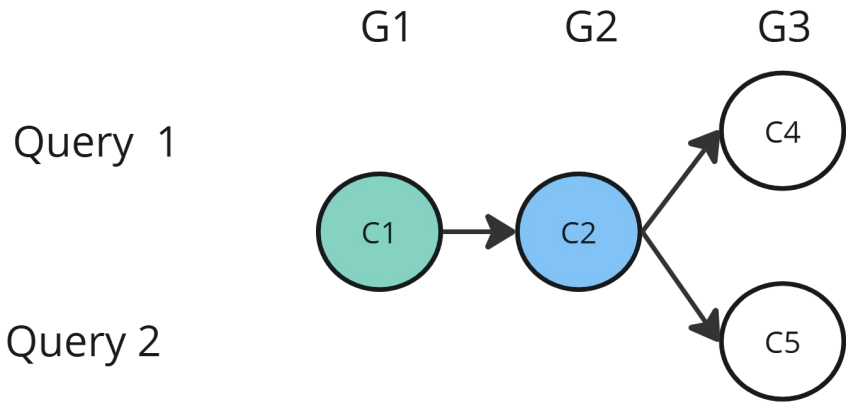
Challenge 1: Redundant transmission of clusters increases latency

Recall C1 and C2 transmitted twice below:



Transmissions for batch of 2 queries

Solution: Transmit each cluster once and compute for all related queries within batch



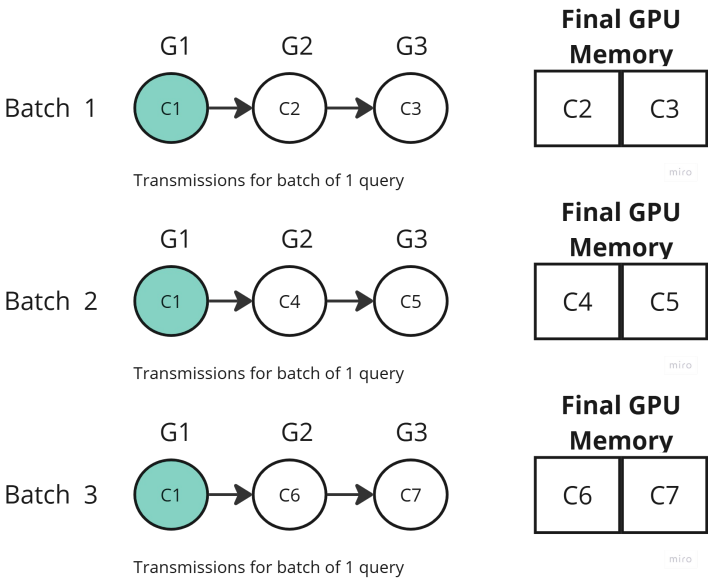
Transmissions for batch of 2 queries

Ci = cluster i
Gi = group j

Memory Manager

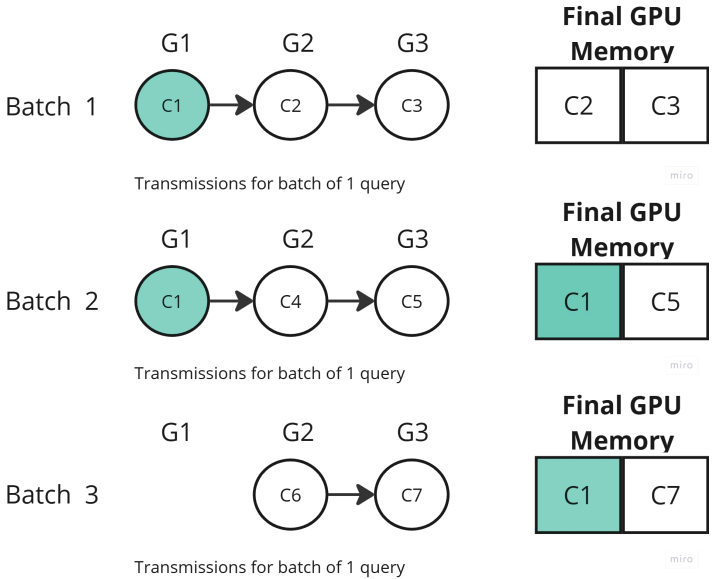
Challenge 1: Redundant transmission of clusters increases latency

How to avoid retransmitting across batches?



Solution: Track how often clusters are used and avoid evicting frequently used clusters

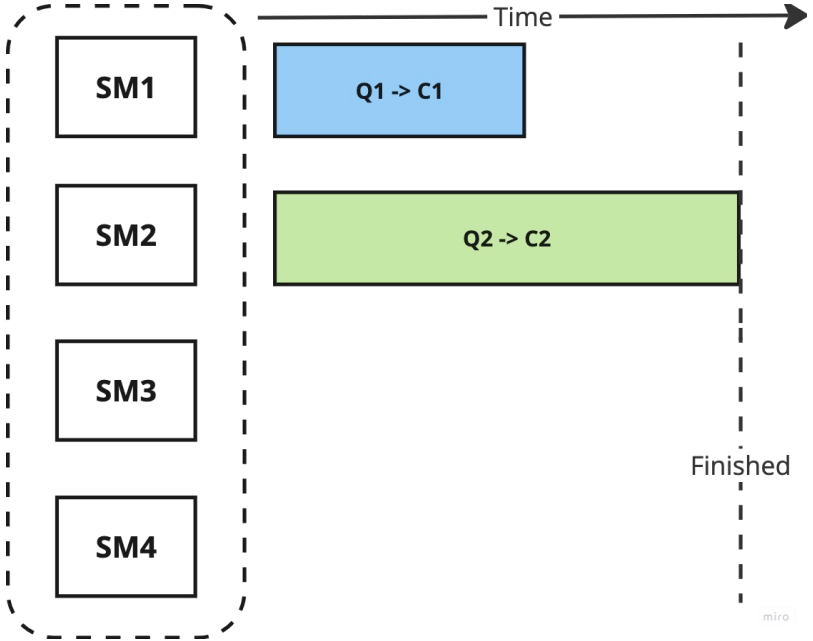
Reuse clusters already in memory



SMi = streaming multiprocessor
Qj = query j
Ck = cluster k
Bm = balanced cluster m

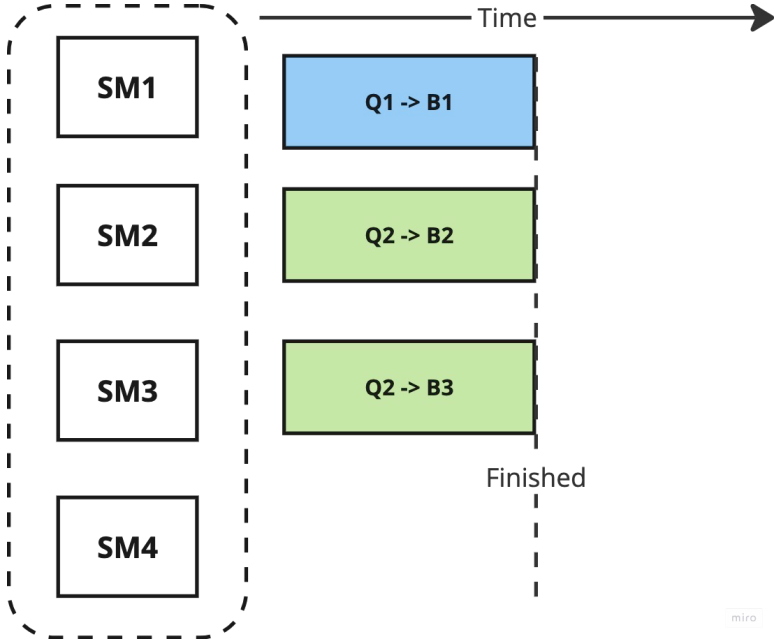
Cluster Balancing

Challenge 2: Some clusters can finish early, leaving them waiting for other clusters (temporal underutilization)



Solution: Split clusters into equally sized chunks (balanced clusters) offline

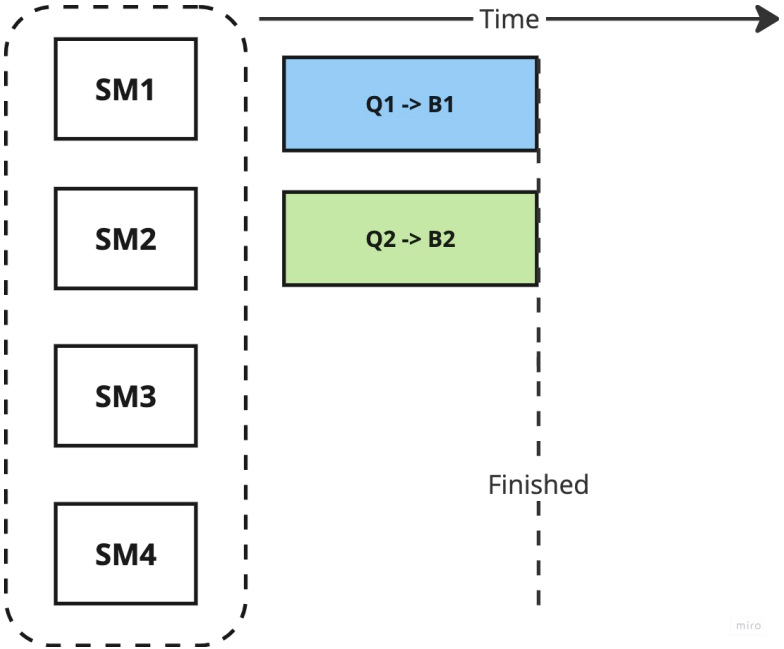
Notice C1 = B1, C2 split into B2 and B3



SMi = streaming multiprocessor
Qj = query j
Ck = cluster k
Bm = balanced cluster m
Tp = thread block p

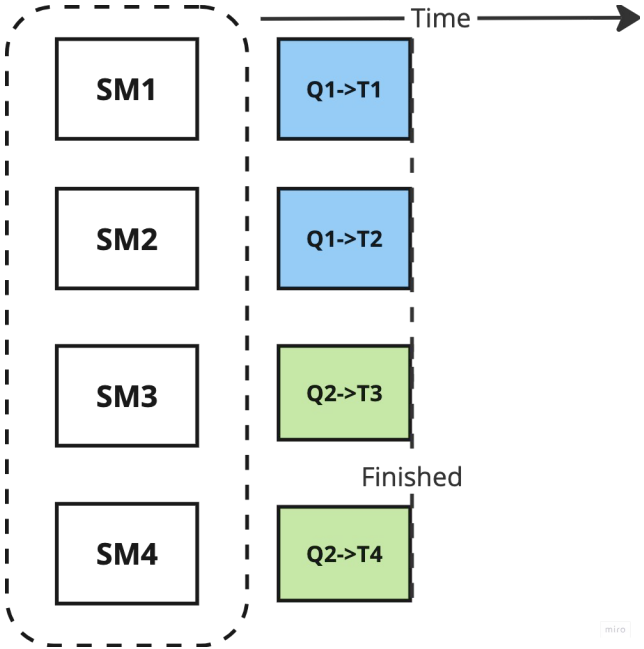
Kernel Controller

Challenge 2: Not always enough clusters to occupy every SM (spatial underutilization)



Solution: Split computation across thread blocks at runtime to occupy all SMs

B1 across T1 and T2, B2 across T3 and T4



Bi = balanced cluster i

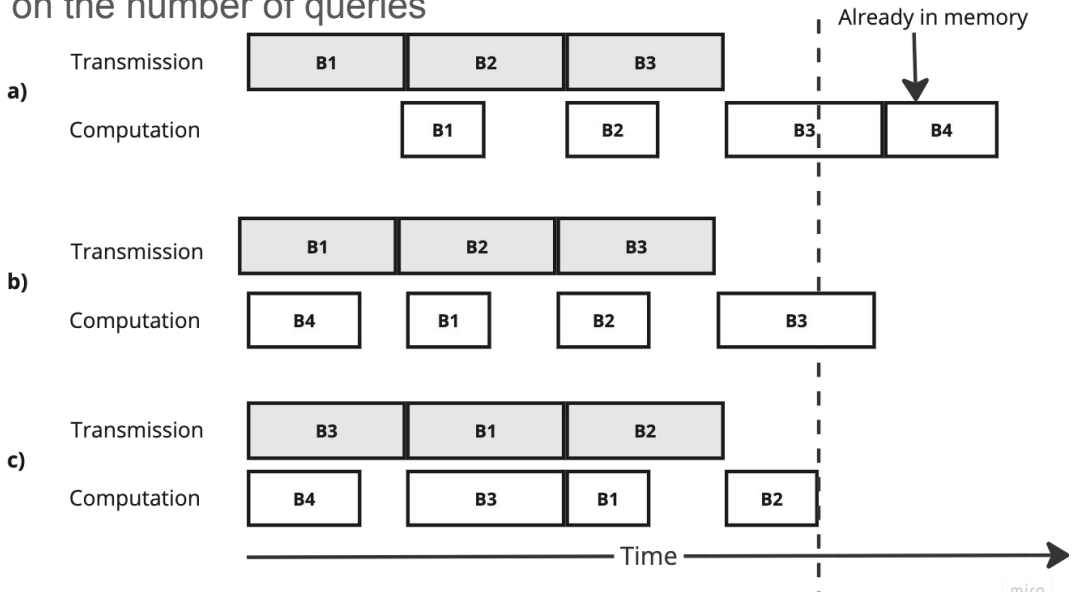
Pipelining Scheduler + Profiler

Challenge 3: Maximizing overlapping of transmission and computation, through ordering, minimizes latency

How to find optimal ordering if computation latencies vary based on the number of queries using a cluster?

Solution: Profile transmission and computation to get latencies

Prioritize longest computations and clusters which are already in GPU



Rummy Overview

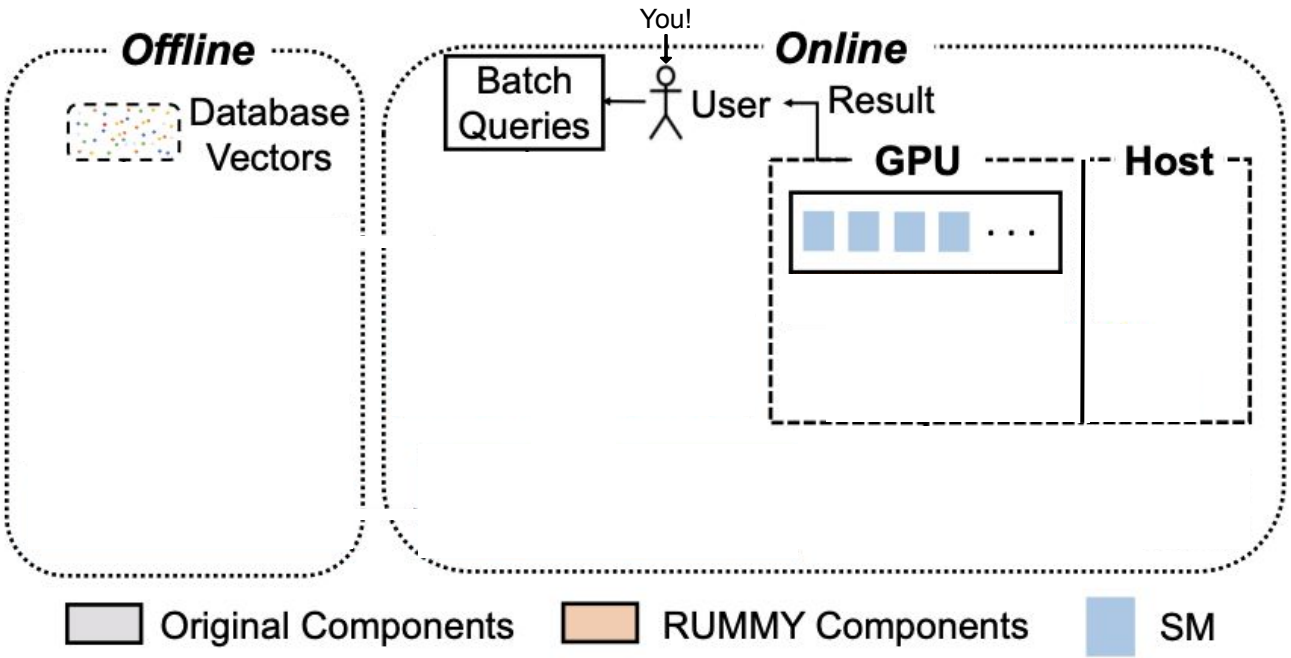


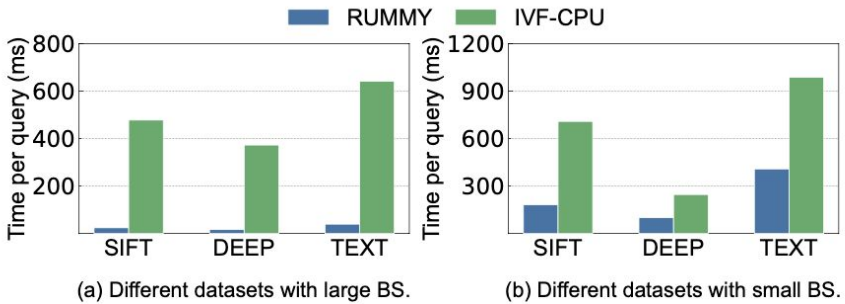
Figure 4: RUMMY overview.

Rummy Evaluation

- Evaluated average end-to-end query latency on AWS (A100 GPU)
- Compared against:
  - Vector query on Intel Xeon E7-8880 CPU
  - Transmitting and computing sequentially on GPU
  - Vector query on GPU using CUDA unified memory
  - Complete overlapping of communication and computation (based on profiler data)
- SIFT1B, DEEP1B, and TEXT1B benchmarks
  - 1 billion database vectors each
  - 10,000-100,000 query vectors each
- Batch sizes (BS) of 2048 (large) or 8 (small) queries

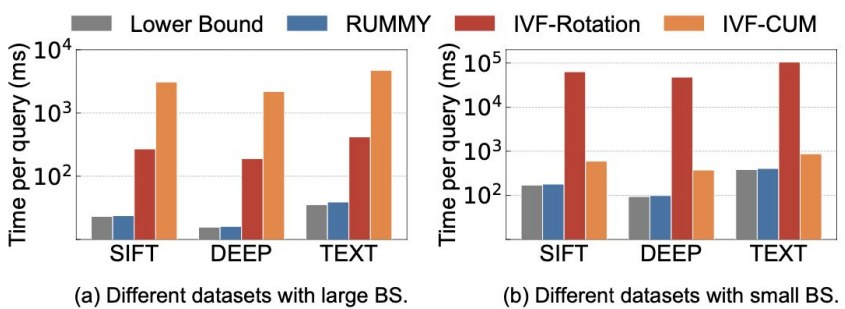
Rummy greatly outperforms existing solutions

- Compared against:
  - Vector query on CPU (IVF-CPU)
  - Transmitting and computing sequentially on GPU (IVF-Rotation)
  - CUDA unified memory (IVF-CUM)
  - Complete overlapping (lower bound)



(a) Different datasets with large BS. (b) Different datasets with small BS.

Figure 10: Comparison between RUMMY and IVF-CPU.



(a) Different datasets with large BS. (b) Different datasets with small BS.

Figure 9: Overall performance on GPUs.

Key Takeaways

Background: Given a query, use vector query to identify relevant documents + send query and documents to LLM

Problem: GPU memory is insufficient for storing large vector query datasets

Idea: Pipeline transmitting clusters of data to and computing on GPU

Solution: Rummy, a GPU vector query system which attempts to…

- Overlap transmission and computation
- Maximize GPU utilization
- Minimize redundant transmission

Backup Slides

Self-RAG

Sample Run: Computer Memory

Input Identify the two types of computer memory. (from `gpt4_alpaca`)

Output [Retrieve=Yes]<p>Computer memory Computer memory – In computing, memory refers to the computer hardware integrated circuits that store information for immediate use in a computer; it is synonymous with the term "primary storage". Computer memory operates at a high speed, for example, random-access memory (RAM), as a distinction from storage that provides slow-to-access information but offers higher capacities. If needed, contents of the computer memory can be transferred to secondary storage; a very common way of doing this is through a memory management technique called "virtual memory". An archaic synonym for memory is store. The term "memory", meaning "primary storage" or "main memory", is<p>[IsRel =Relevant]Primary memory, also known as main memory, is the memory that the computer can directly access and manipulate. [IsSup =Fully Supported] [Retrieve=Yes]<p>Computer memory–often associated with addressable semiconductor memory, i.e. integrated circuits consisting of silicon-based transistors, used for example as primary storage but also other purposes in computers and other digital electronic devices. There are two main kinds of semiconductor memory, volatile and non-volatile. Examples of non-volatile memory are flash memory (used as secondary memory) and ROM, PROM, EPROM and EEPROM memory (used for storing firmware such as BIOS). Examples of volatile memory are primary storage, which is typically dynamic random-access memory (DRAM), and fast CPU cache memory, which is typically static random-access memory (SRAM) that is fast but energy-consuming, offering lower memory<p>[IsRel =Relevant]Primary memory can be accessed quickly, but it is volatile, meaning that the data is lost if the power is turned off.[IsSup =Partially Supported]...[IsUse :5]

Sample Run: Poem + Walking Dead

Input Synthesize a poem based on this prompt:"The sun is setting" (from `stanford_alpaca`)

Output [Retrieve=No]The stars begin to twinkle bright the trees, scattered throughout the night; [Retrieve=No]The stars begin to twinkle bright the trees, scattered throughout the night [Retrieve=No] They flicker like a silent dance, A peaceful moment in its trance. [Retrieve=No] They flicker like a silent dance, A peaceful moment in its trance. [Retrieve=No] A chorus of crickets sing a lullaby, To every creature that roams the sky [ISUSE =5]

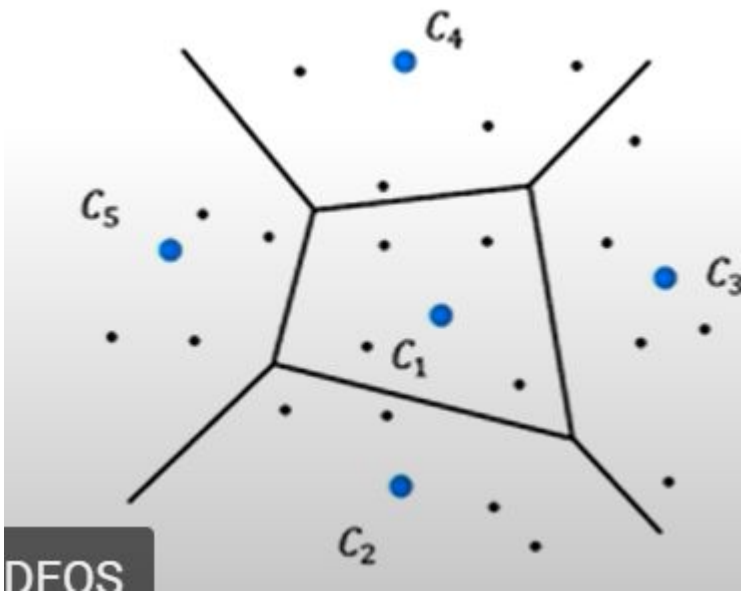
Input when did walking dead season 7 come out (from `nq`)

Output [Retrieve=Yes]<p>The Walking Dead (season 7)–The Walking Dead (season 7) The seventh season of "The Walking Dead", an American post-apocalyptic horror television series on AMC, premiered on October 23, 2016, and concluded on April 2, 2017, consisting of 16 episodes. Developed for television by Frank Darabont, the series is based on the eponymous series of comic books by Robert Kirkman, Tony Moore, and Charlie Adlard. ...<p>[ISREL =Relevant]October 23 , 2016[ISSUP =Fully Supported][ISUSE =5]

Rummy

Vector Query Implementation

- Top-k nearest neighbor (KNN) finds *exact* top-k results most similar to query vector
- Approximate top-k nearest neighbor (ANN)
  - Graph Index
  - Inverted File Index



Challenges with Pipelining

Cross-query redundant transmission

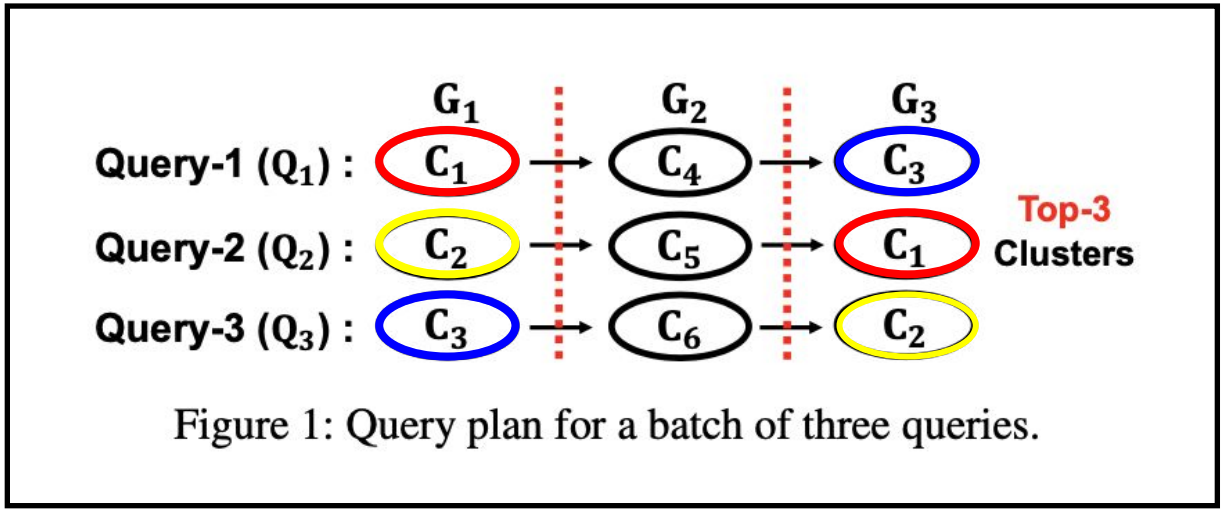


Figure 1: Query plan for a batch of three queries.

Challenges with Pipelining

Maximizing overlap between transmission and computation

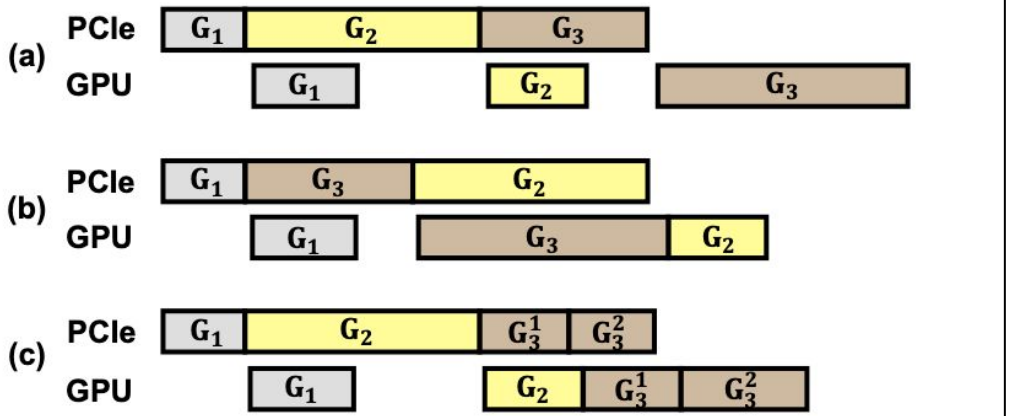
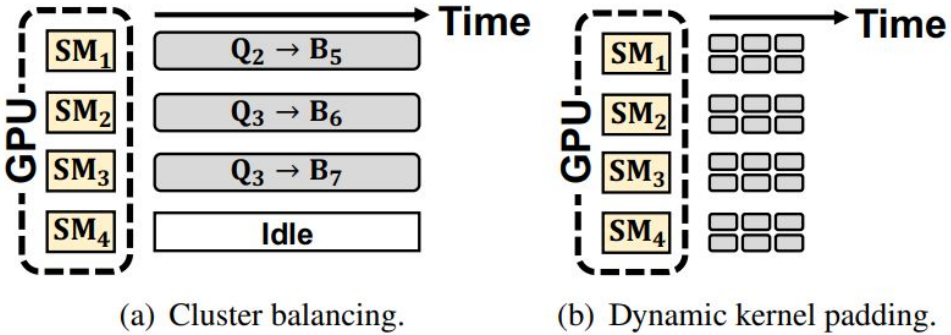


Figure 2: Examples for the pipelining plans (ignore data reuse).

Rummy - Spatial and temporal GPU underutilization

Cluster Balancing - makes clusters, and thus computation time, the same size

Kernel Controller - splits computation for each cluster across multiple SMs



(a) Cluster balancing. (b) Dynamic kernel padding.

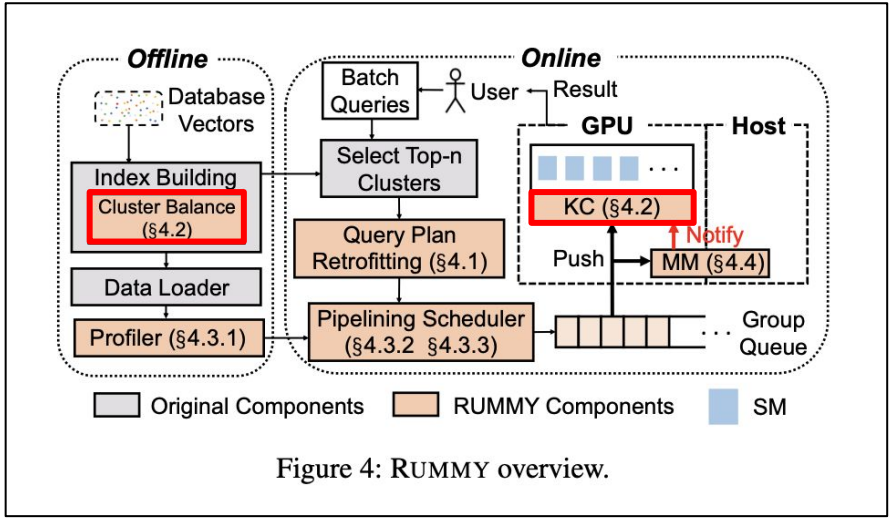


Figure 4: RUMMY overview.

Rummy - Cross-query redundant transmission

Query Plan Retrofitting - keep clusters in GPU until all queries are done with them

Memory management - avoids evicting frequently used clusters and uses paging

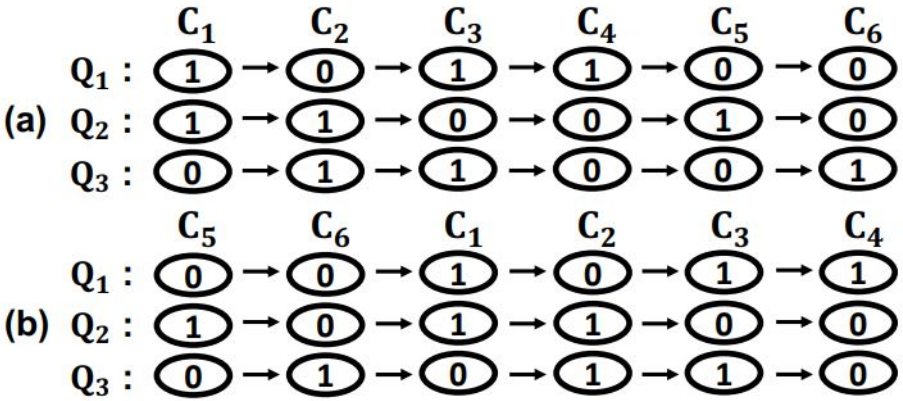


Figure 5: Retrofit the vector query plan.

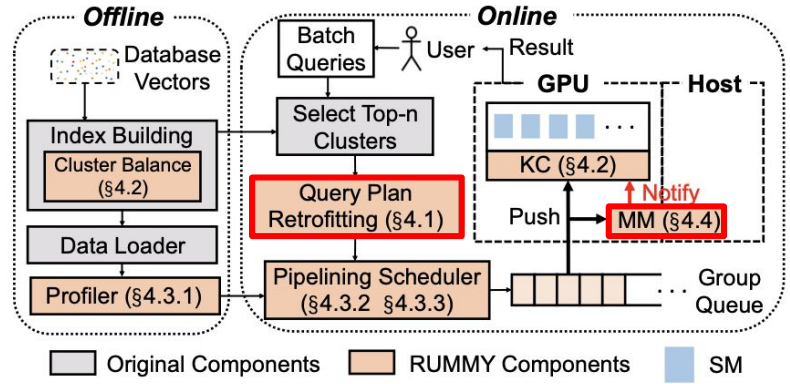


Figure 4: RUMMY overview.

Rummy - Transmission and computation overlapping

Profiler - Measures transmission and computation time for each group

Pipelining scheduler - Does longer computations and computations for clusters already in memory early

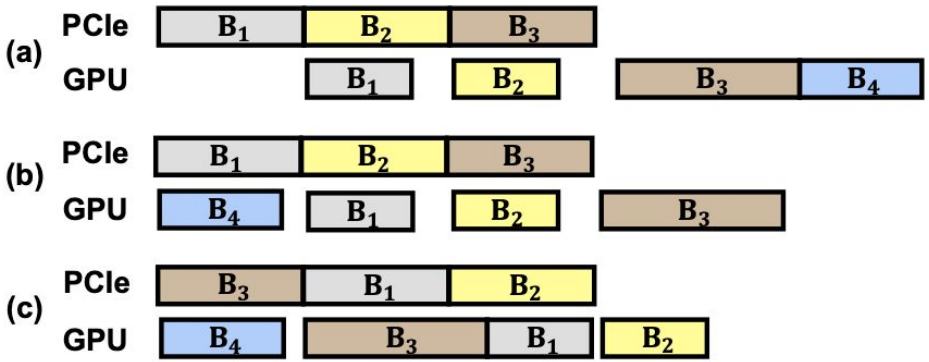


Figure 8: Examples for the reordering algorithm.

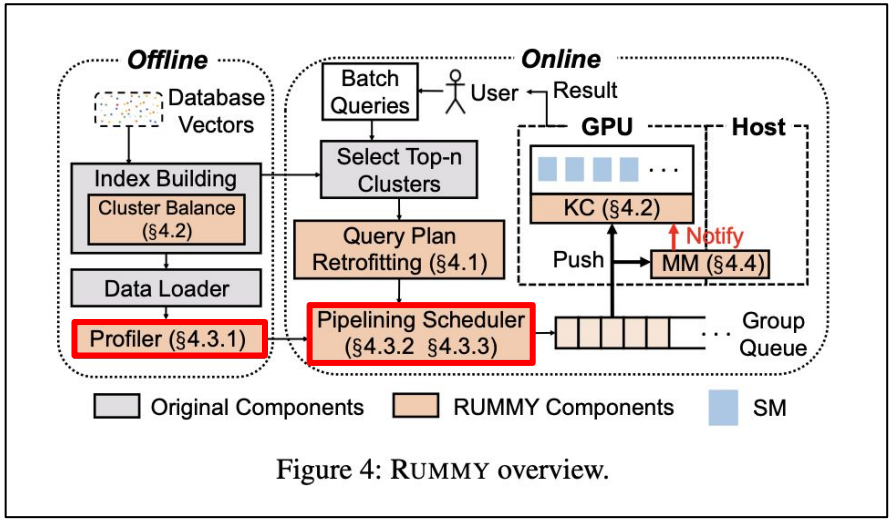


Figure 4: RUMMY overview.

Rummy Overview

“GPU-accelerated vector query processing system to support large vector datasets beyond GPU memory”

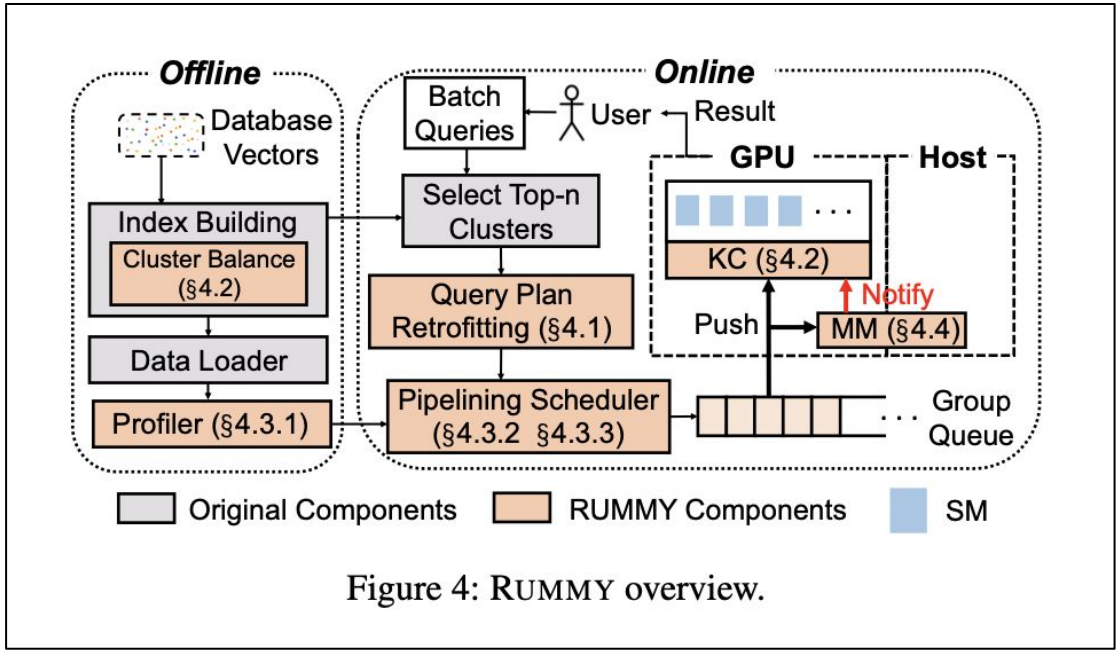


Figure 4: RUMMY overview.

Rummy's dynamic kernel padding with cluster balancing

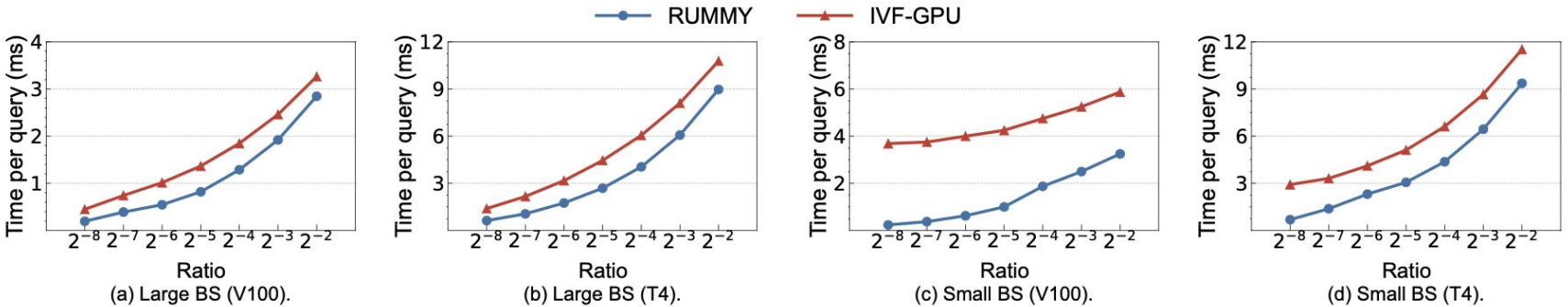


Figure 12: Effectiveness of RUMMY's dynamic kernel padding with cluster balancing.

- Doesn't consider transmission to isolate results
- Rummy outperforms IVF-GPU (1.2-2.3x) due to cluster balancing, which mitigates straggler block in kernel

Effectiveness of Rummy's profiler

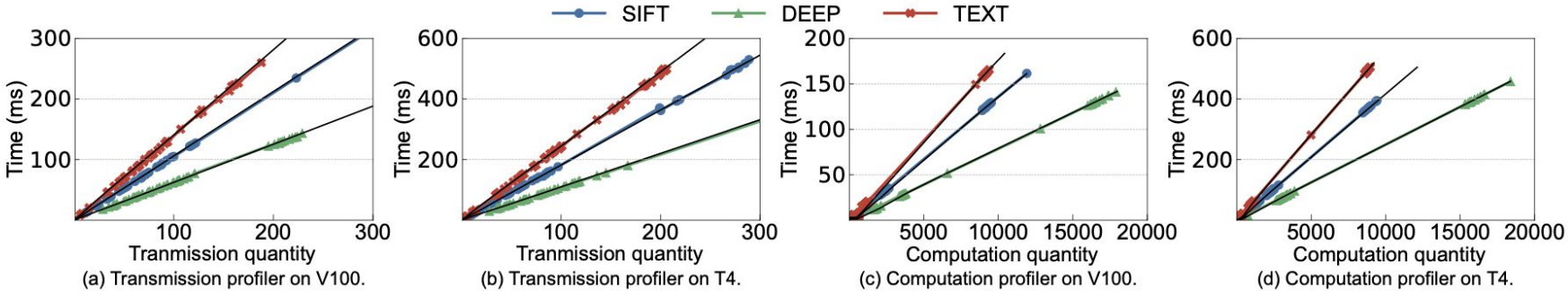


Figure 13: Effectiveness of RUMMY's profiler.

- Black lines show profiler's estimated time, while colored lines depict actual runtime

Rummy's reordering and grouping with cluster-based retrofitting

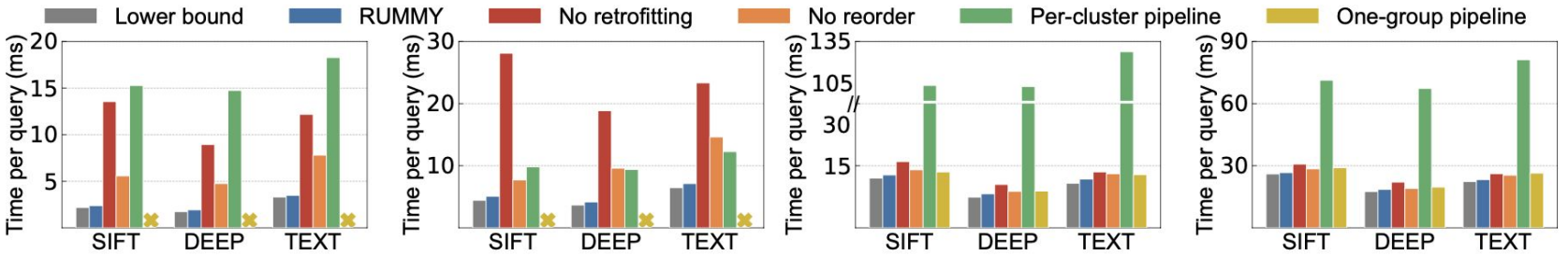


Figure 14: Effectiveness of RUMMY's reordering and grouping with cluster-based retrofitting.

- Rummy achieves closest performance to lower bound

Rummy's GPU memory management

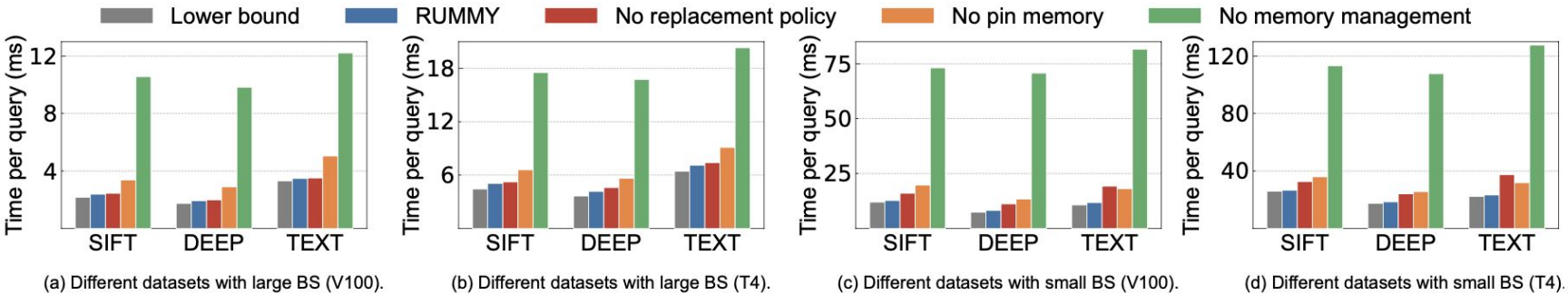


Figure 15: Effectiveness of RUMMY's GPU memory management.

- All memory management techniques are effective
- Combination of techniques will yield lowest query time

Original RAG image