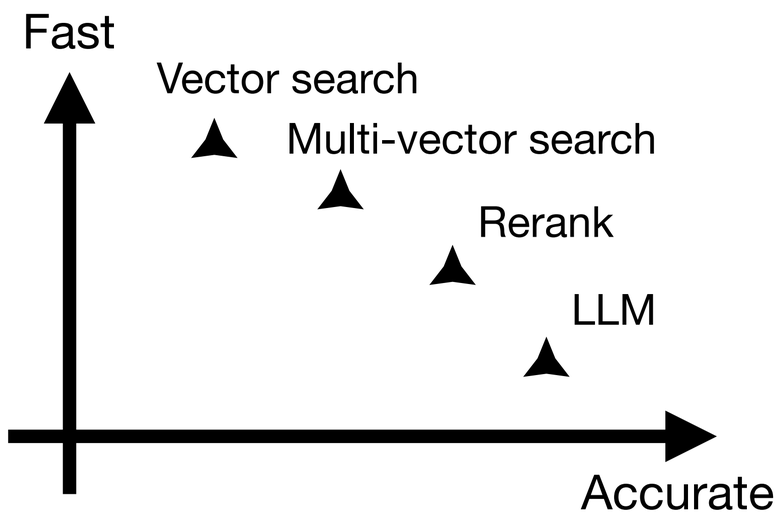
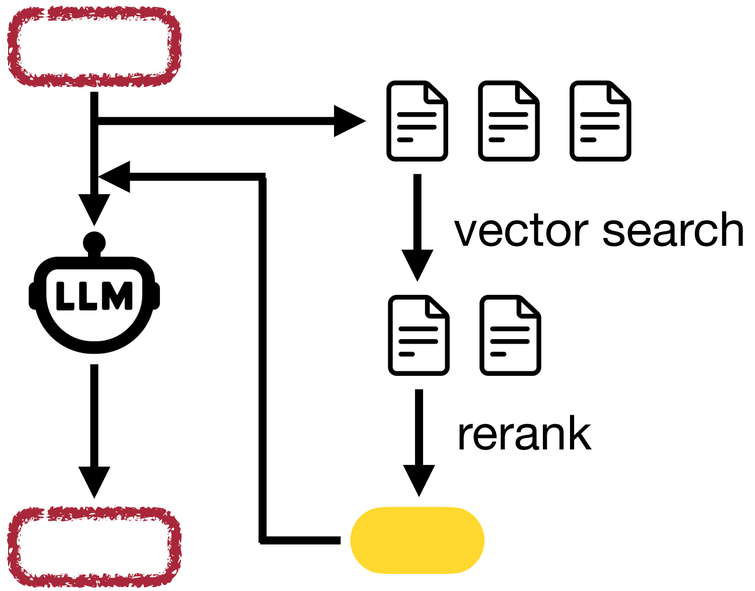
Towards
Self-designing RAG Systems

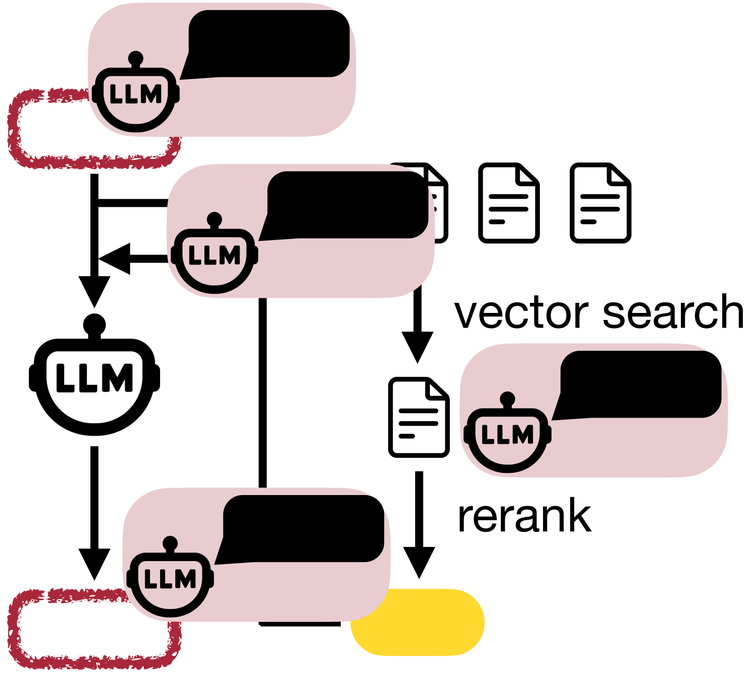
Qitong Wang, qitong@seas.harvard.edu

CS265. February 5, 2026

DASlab
@ Harvard SEAS

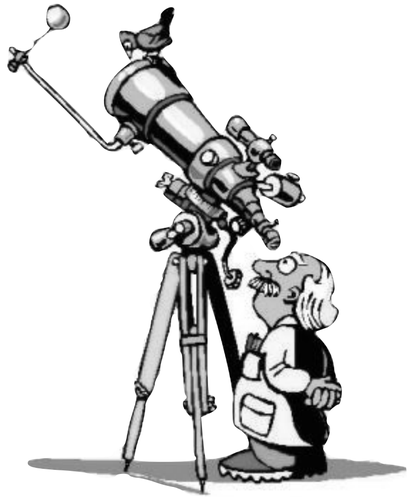






- Data systems vs. RAG systems
  Storage, compute
- System designs
  Resource allocation, pipelining, configuration
- Self-designing RAG systems

This class



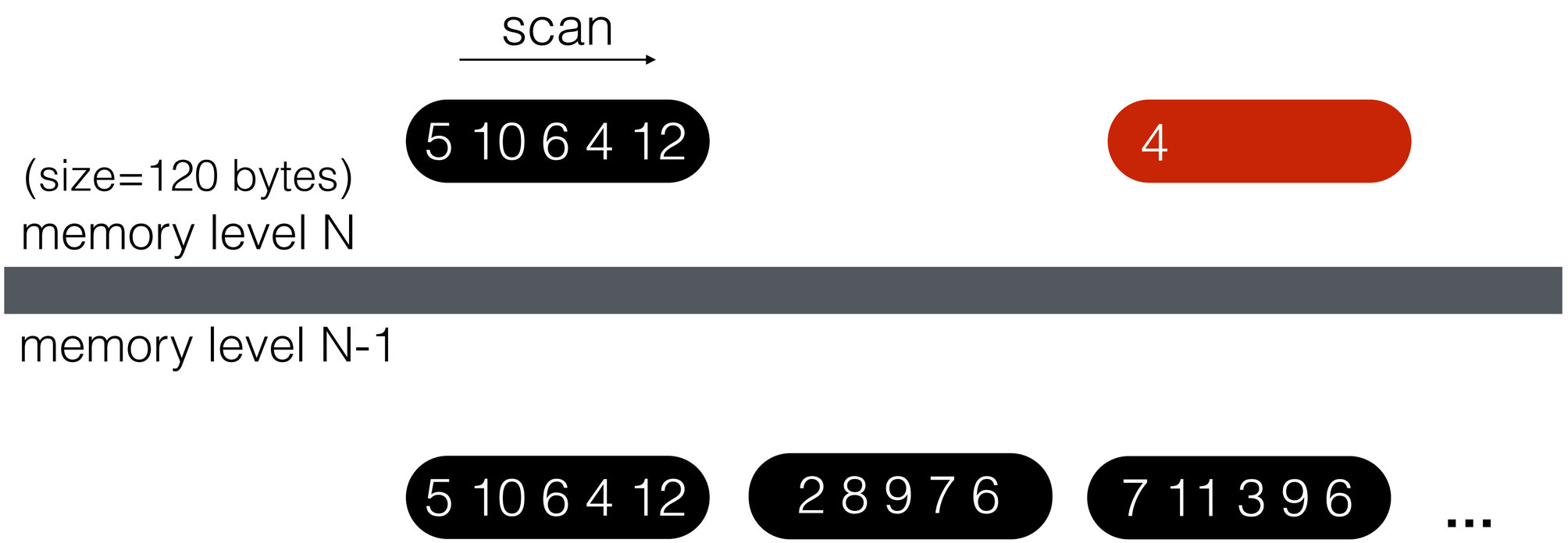
data

star(id, name, distance, density, …)

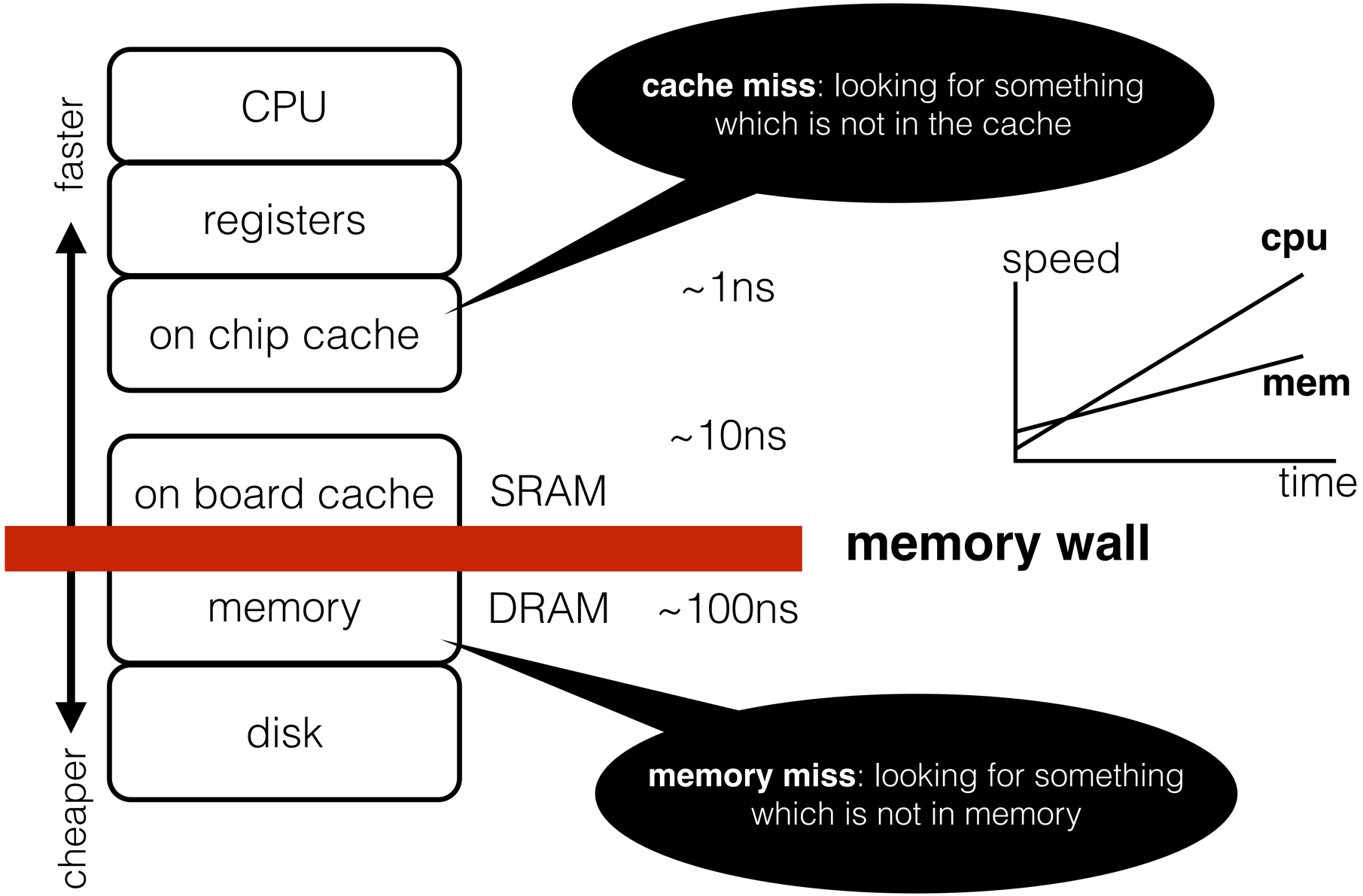
[1, star1, x1, y1, …]
[2, star2, x2, y2, …]
[3, star3, x3, y3, …]
[4, star4, x4, y4, …]
…

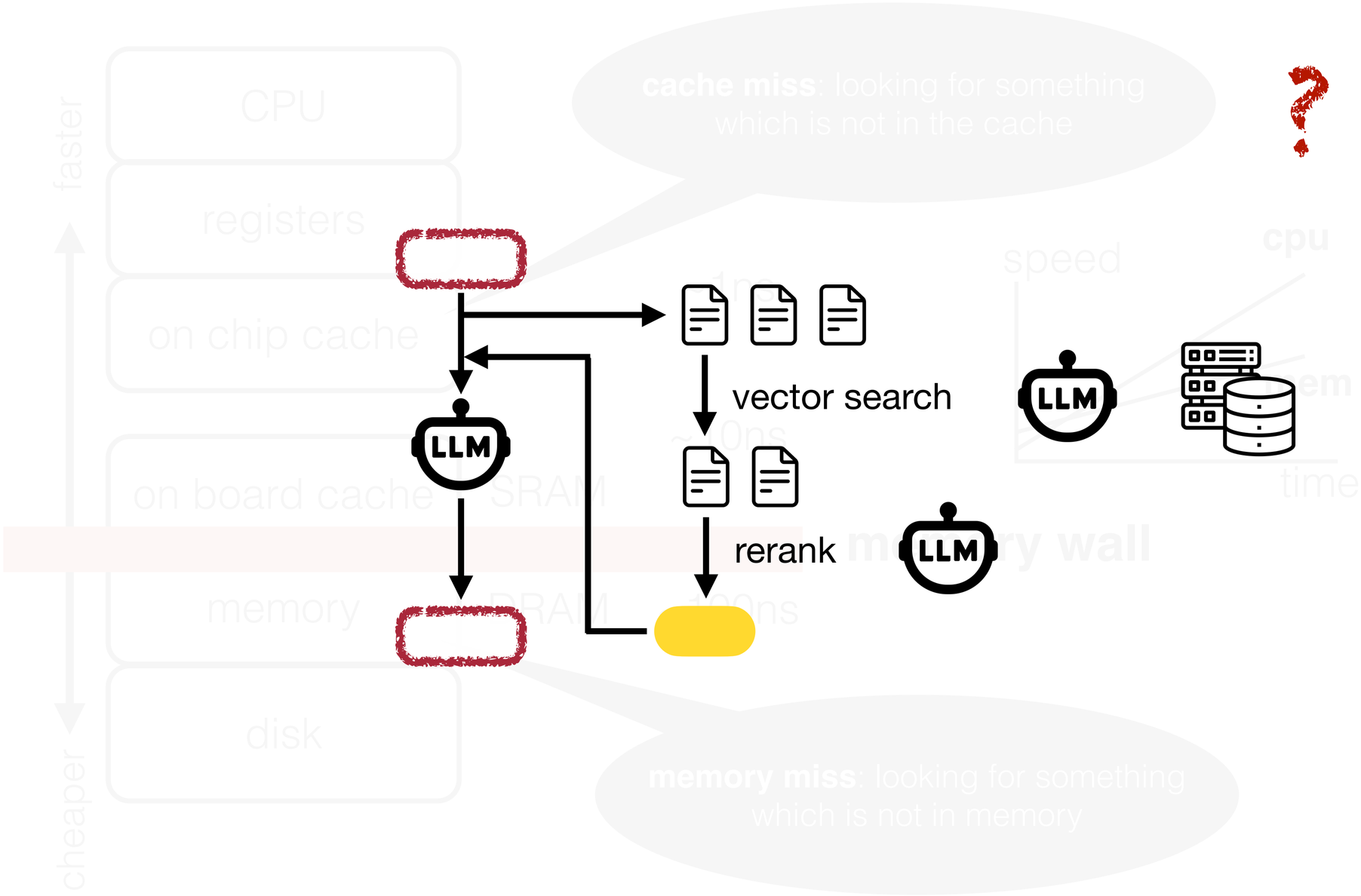
data size	store - access	
10s/100s	paper - just look at it!	**data collection is the key**
K/M	files - shell/excel	**learn a bit how computers work**
B	files - custom	**need serious programming skills**
T	data sys. - declarative	**data-driven analysis**

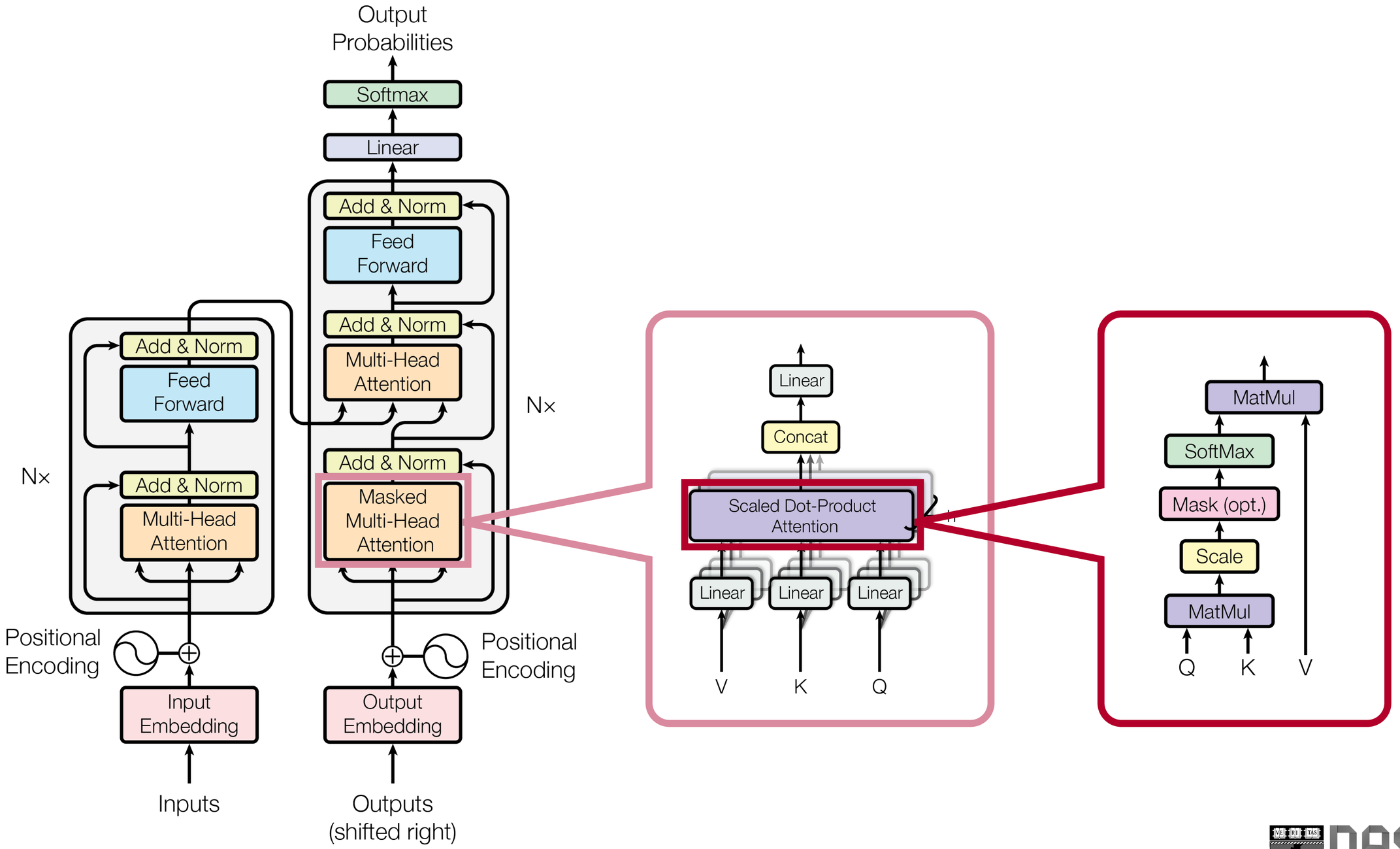
query x<5



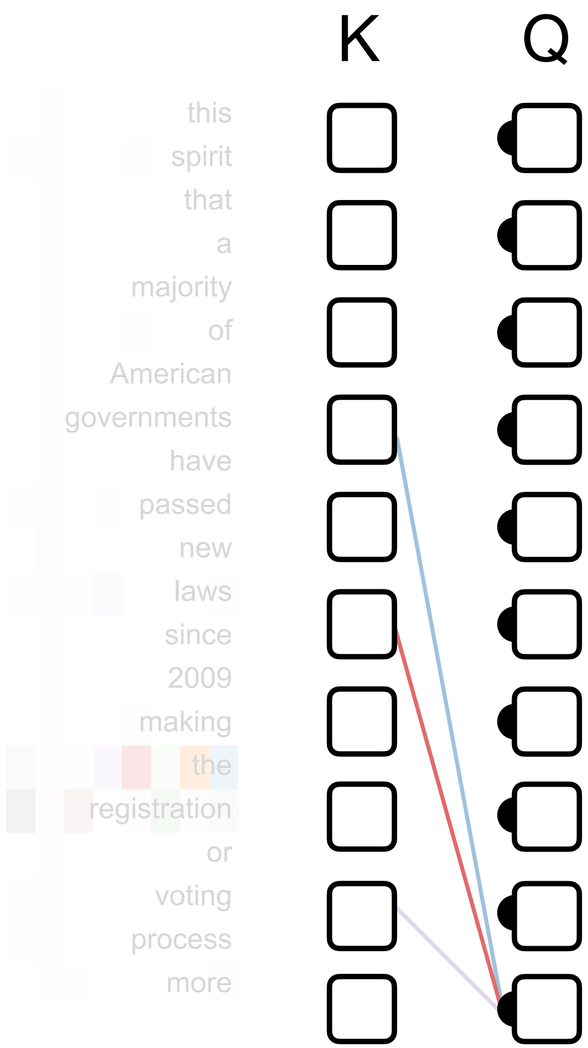
page size: 5x8 bytes


CPU
registers
on chip cache
on board cache
memory
disk
faster
cheaper
~1ns
~10ns
SRAM
DRAM
~100ns
cache miss: looking for something which is not in the cache
memory miss: looking for something which is not in memory
memory wall
speed
cpu
mem
time


cache miss: looking for something which is not in the cache
CPU
registers
on chip cache
on board cache
SRAM
memory
DRAM
disk
faster
cheaper
LLM
vector search
rerank
LLM
LLM
memory miss: looking for something which is not in memory

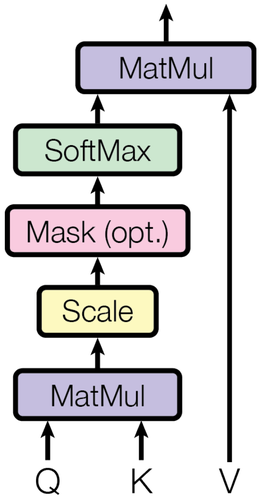


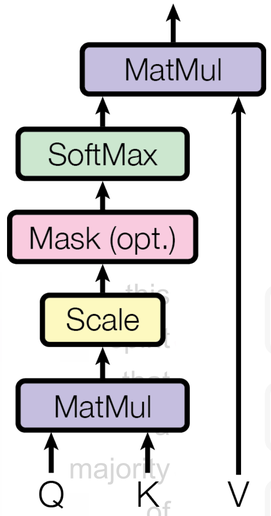
Ashish Vaswani, et al. Attention is All you Need. NIPS 2017



A semantic vector

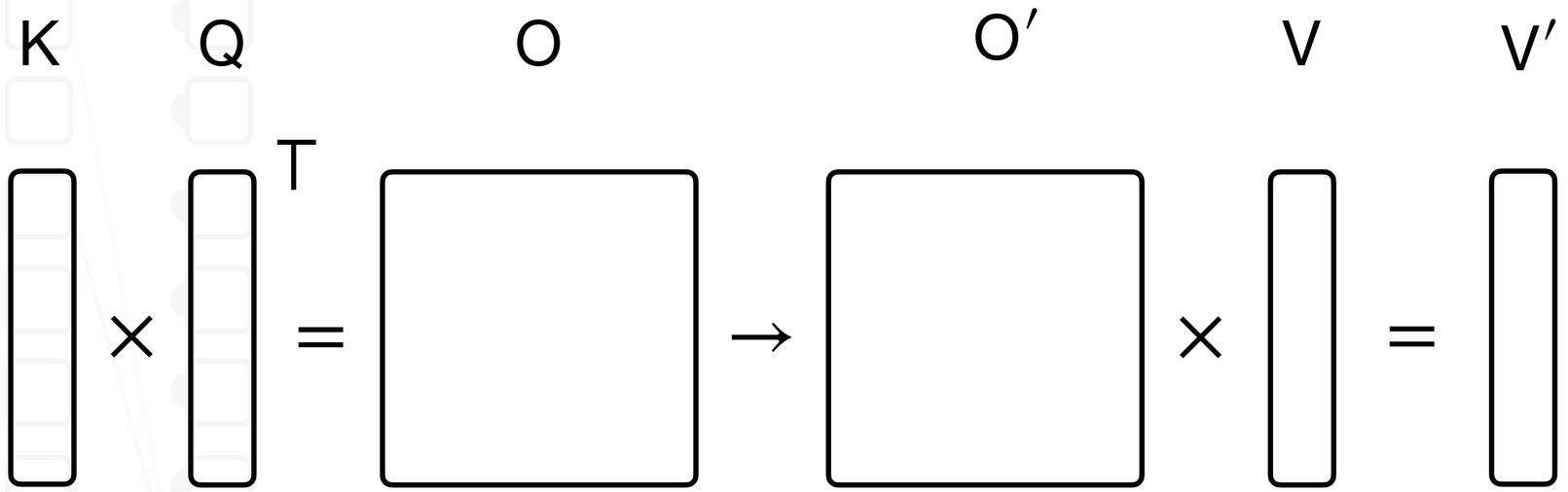
[5.4, 1.7, 4.7, 3.4, …, 5.3, 5.1, 8.1, 6.9]

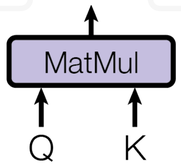


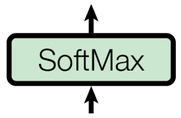


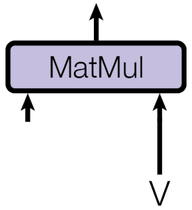
A semantic vector

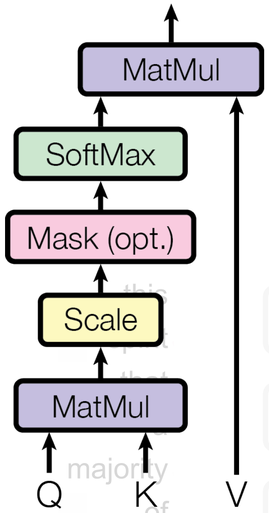
[5.4, 1.7, 4.7, 3.4, …, 5.3, 5.1, 8.1, 6.9]

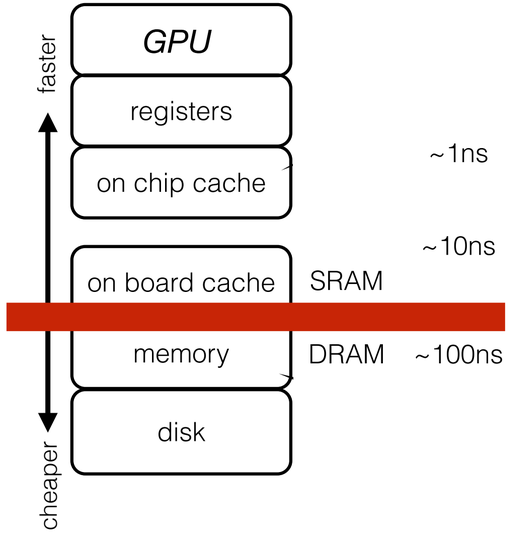


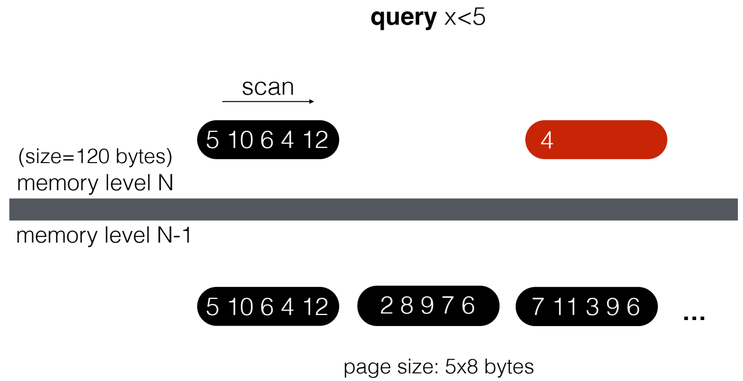




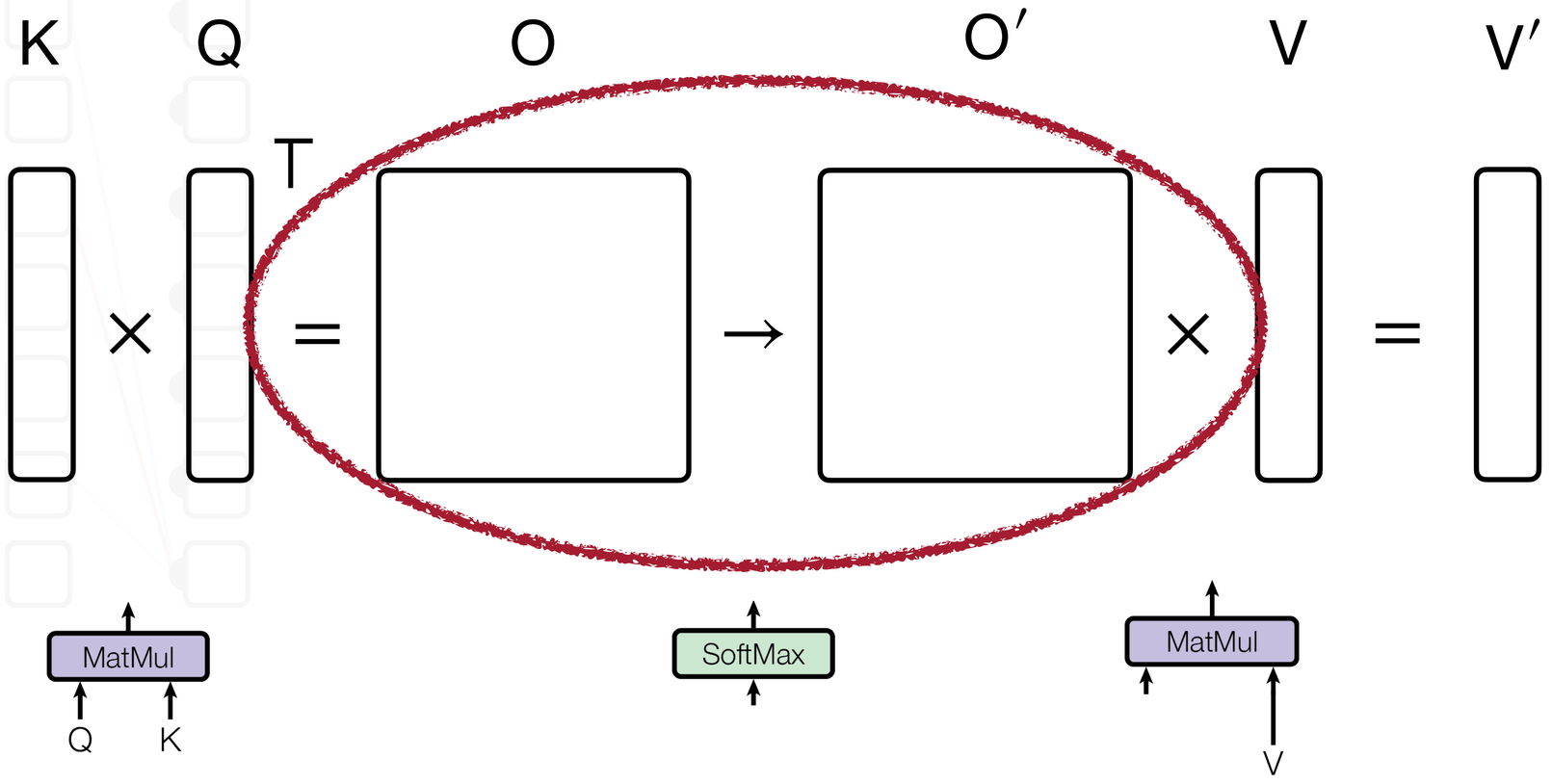


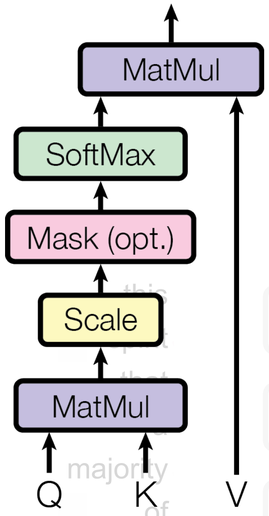


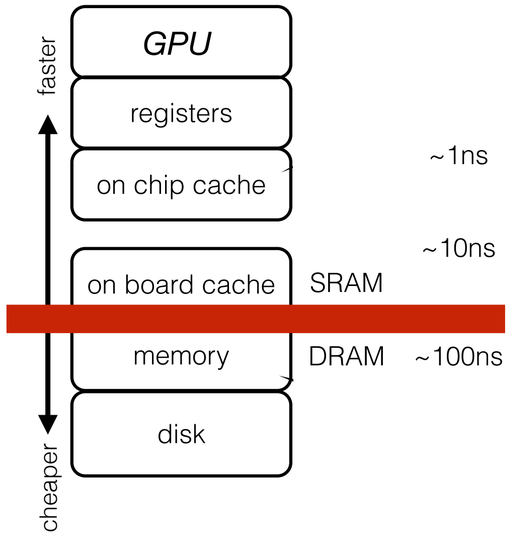


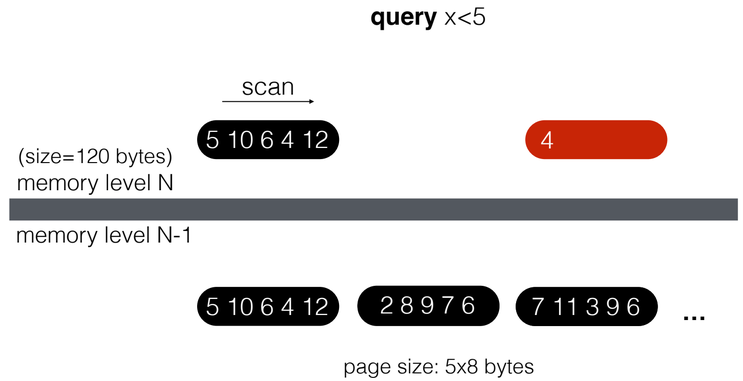


Compared to data systems?

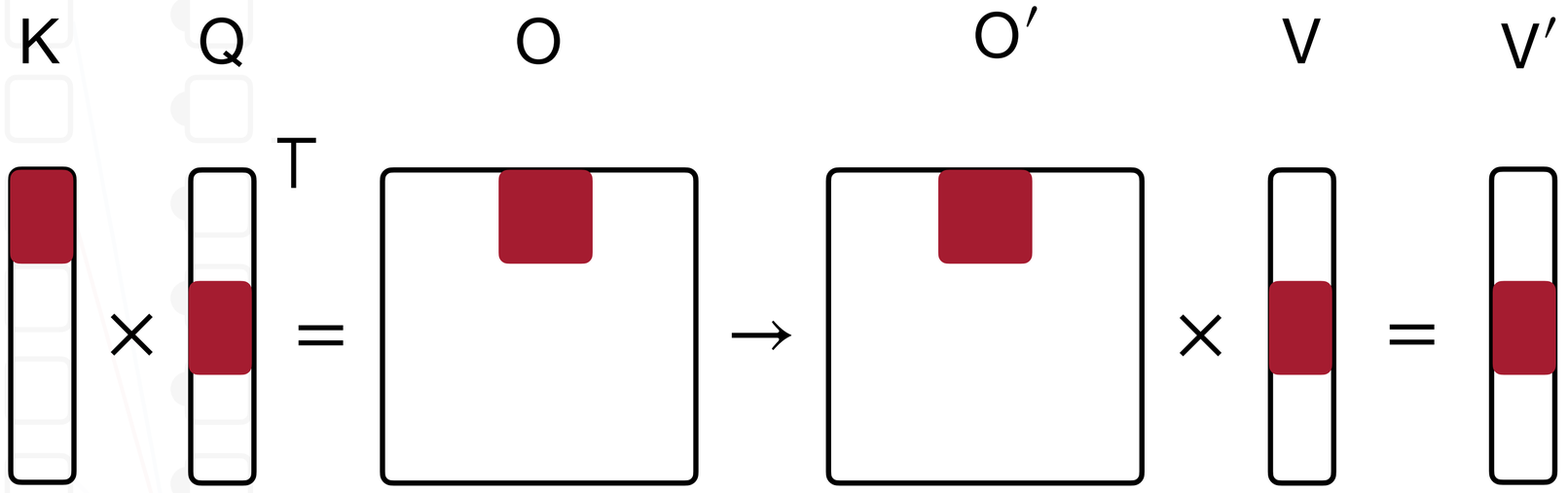


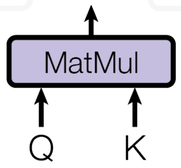


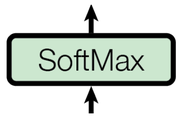


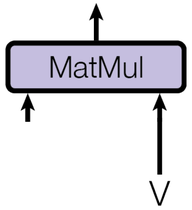


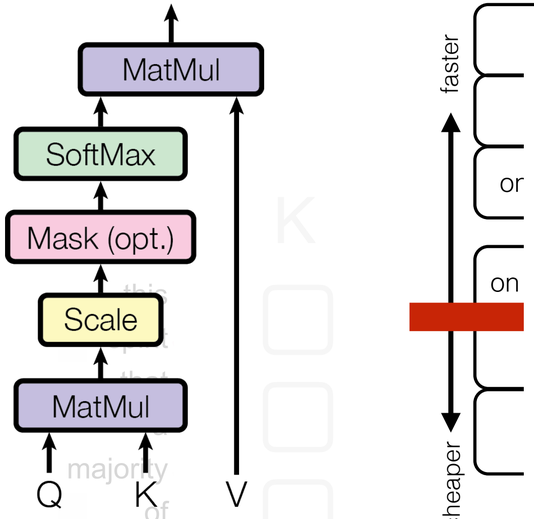
Compared to data systems?

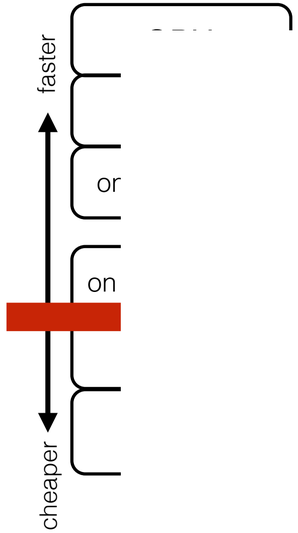


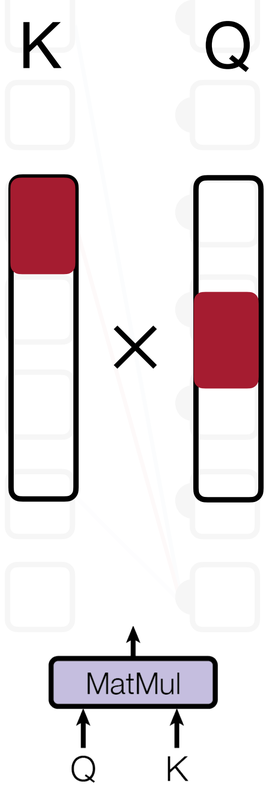












majority of American governments have passed new laws since 2009 making the registration or voting process more

FLASHATTENTION: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao[†], **Daniel Y. Fu** [†], **Stefano Ermon** [†], **Atri Rudra** [‡], **Christopher Ré** [†]
[†] Department of Computer Science, Stanford University
[‡] Department of Computer Science and Engineering, University at Buffalo, SUNY
{trid,danfu}@stanford.edu, ermon@stanford.edu, atri@buffalo.edu, chrismre@cs.stanford.edu

Abstract

Transformers are slow and memory-hungry on long sequences, since the time and memory complexity of self-attention are quadratic in sequence length. Approximate attention methods have attempted to address this problem by trading off model quality to reduce the compute complexity, but often do not achieve wall-clock speedup. We argue that a missing principle is making attention algorithms *IO-aware*—accounting for reads and writes between levels of GPU memory. We propose FLASHATTENTION, an IO-aware exact attention algorithm that uses tiling to reduce the number of memory reads/writes between GPU high bandwidth memory (HBM) and GPU on-chip SRAM. We analyze the IO complexity of FLASHATTENTION, showing that it requires fewer HBM accesses than standard attention, and is optimal for a range of SRAM sizes. We also extend FLASHATTENTION to block-sparse attention, yielding an approximate attention algorithm that is faster than any existing approximate attention method. FLASHATTENTION trains Transformers faster than existing baselines: 15% end-to-end wall-clock speedup on BERT-large (seq. length 512) compared to the MLPerf 1.1 training speed record, 3× speedup on GPT-2 (seq. length 1K), and 2.4× speedup on long-range arena (seq. length 1K-4K). FLASHATTENTION and block-sparse FLASHATTENTION enable longer context in Transformers, yielding higher quality models (0.7 better perplexity on GPT-2 and 6.4 points of lift on long-document classification) and entirely new capabilities: the first Transformers to achieve better-than-chance performance on the Path-X challenge (seq. length 16K, 61.4% accuracy) and Path-256 (seq. length 64K, 63.1% accuracy).

1 Introduction

Transformer models [86] have emerged as the most widely used architecture in applications such as natural language processing and image classification. Transformers have grown larger [5] and deeper [87], but equipping them with longer context remains difficult [83], since the self-attention module at their heart has time and memory complexity quadratic in sequence length. An important question is whether making attention faster and more memory-efficient can help Transformer models address their runtime and memory challenges for long sequences.

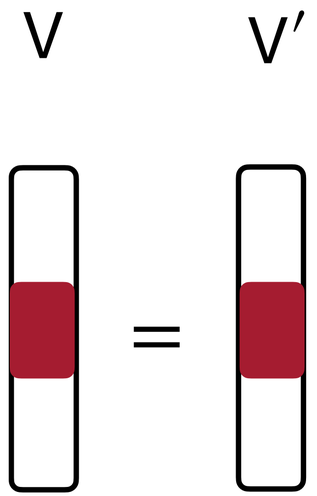
Many approximate attention methods have aimed to reduce the compute and memory requirements of attention. These methods range from sparse-approximation [53, 77] to low-rank approximation [13, 52, 88], and their combinations [3, 9, 96]. Although these methods reduce the compute requirements to linear or near-linear in sequence length, many of them do not display wall-clock speedup against standard attention and have not gained wide adoption. One main reason is that they focus on FLOP reduction (which may not correlate with wall-clock speed) and tend to ignore overheads from memory access (IO).

In this paper, we argue that a missing principle is making attention algorithms *IO-aware* [1]—that is, carefully accounting for reads and writes to different levels of fast and slow memory (e.g., between fast GPU on-chip SRAM and relatively slow GPU high bandwidth memory, or HBM [47], Figure 1

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

4

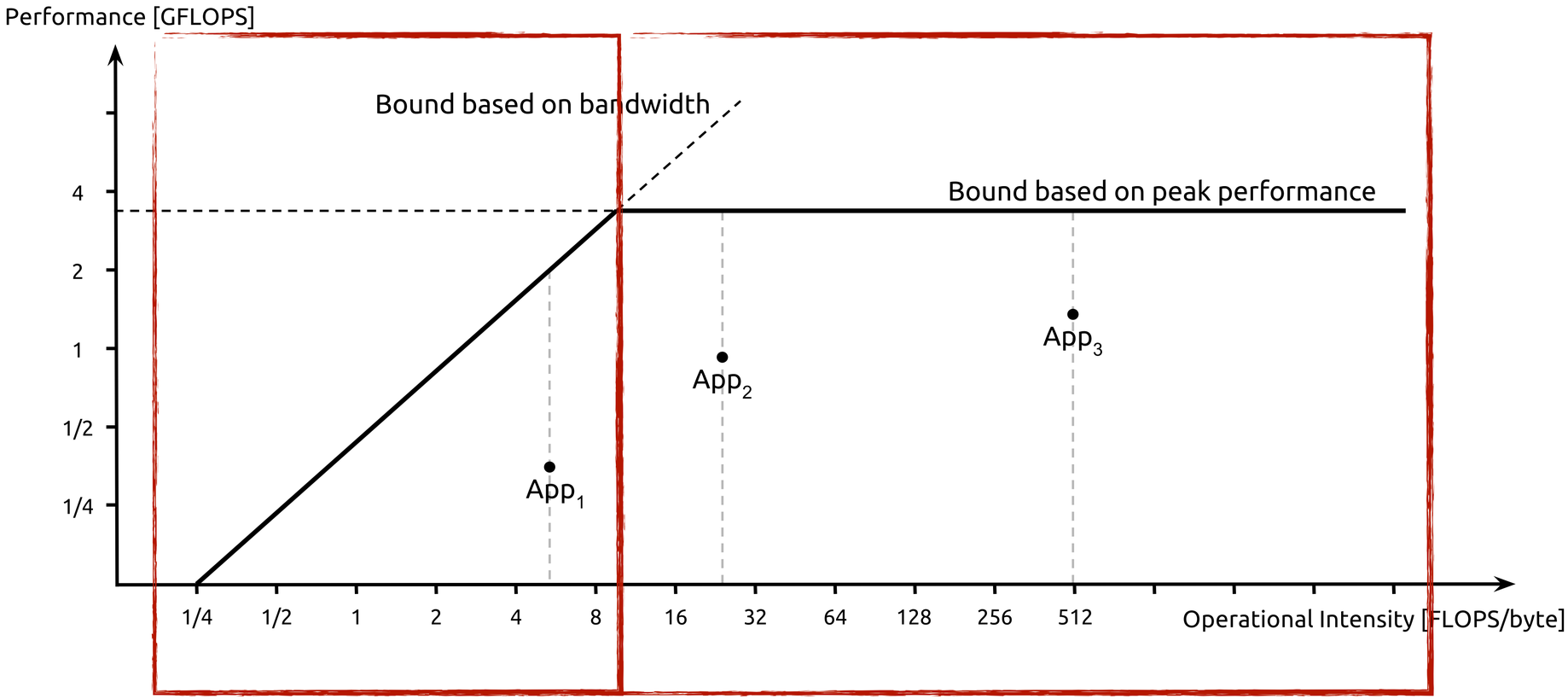
7 11 3 9 6 ...



Compared to data systems?

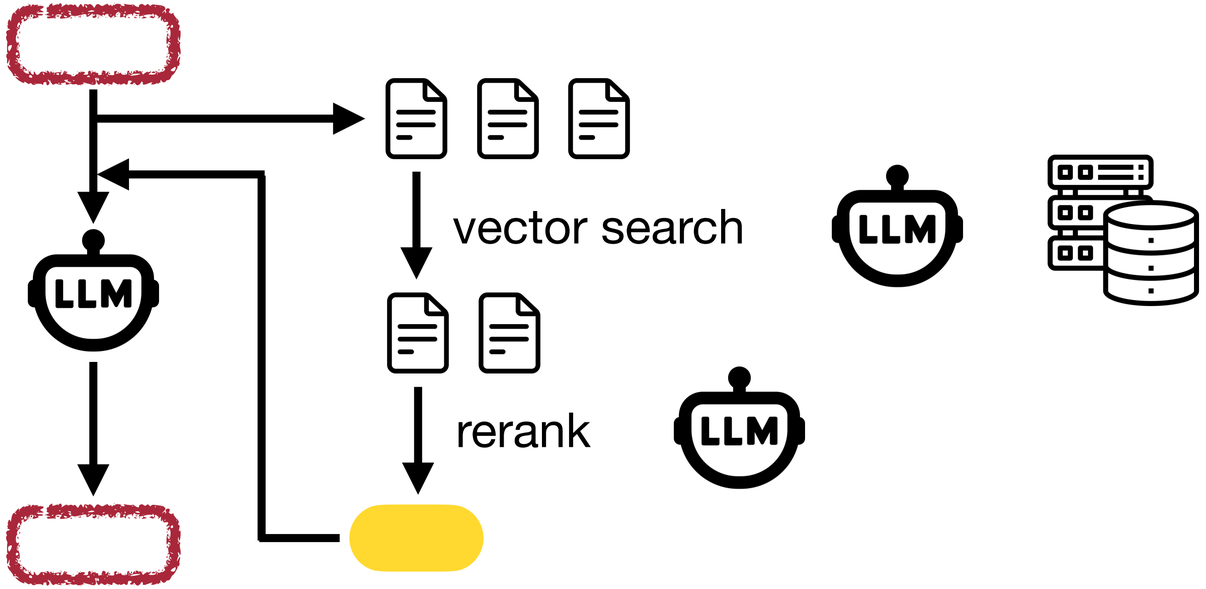
compute / data = x ops / y bytes := **arithmetic intensity**

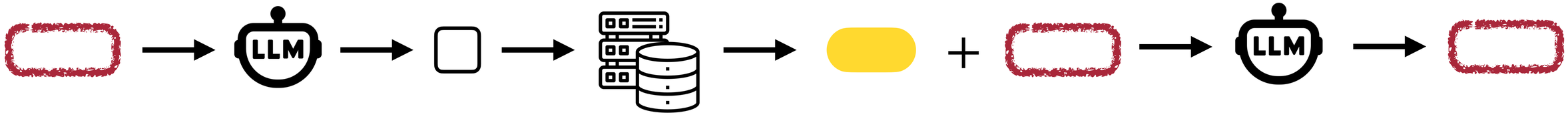
compute is the bottleneck



The roofline model

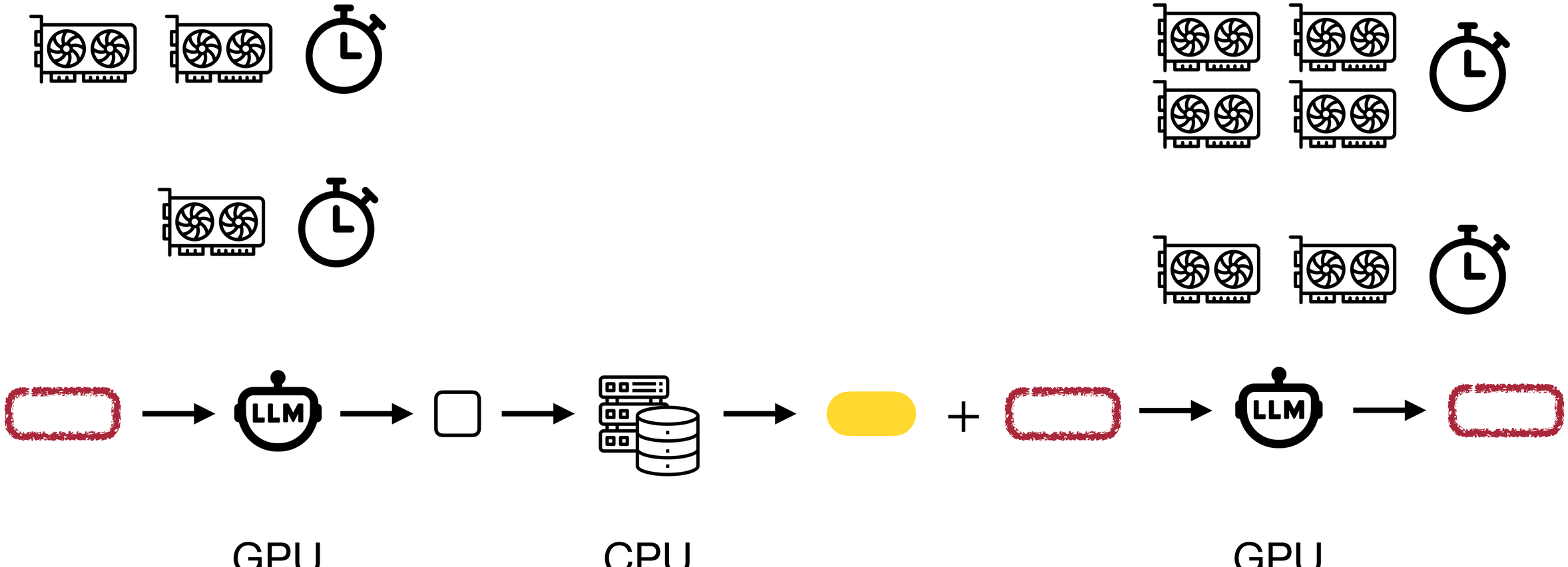
data movement (I/O) is the bottleneck





GPU CPU GPU

Resource allocation



TorchSim: High Fidelity Runtime and Memory Estimation for Distributed Training

Sanket Purandare [1 2] **Emma Yang** [1] **Andrew Zhao** [1] **Qitong Wang** [1] **Wei Feng** [2] **Alban Desmaison** [2]
Andrew Gu [2] **Tianyu Liu** [2] **Less Wright** [2] **Gokul Nadathur** [2] **Stratos Idreos** [1]

Abstract

Large AI models unlock powerful applications but are costly and complex to train, primarily due to the challenge of configuring distributed training across GPU clusters. This involves selecting the right combination of techniques based on the model, data, hardware, and performance objectives. In practice, teams often rely on trial and error, leading to high compute costs, cloud spend, and wasted time, without guarantees of success or optimality. We present TorchSim, a simulator that eliminates this burden by accurately predicting whether a configuration will succeed (i.e., stay within memory limits) and how long it will take to run, without requiring actual execution or access to the target hardware. Users simply input candidate configurations and choose the best successful one, such as the fastest, avoiding costly and uncertain tuning. TorchSim combines analytical and learned models to estimate operator-level runtimes and employs a GPU execution simulator to capture the intricacies of multi-stream parallelism and hardware behavior. Evaluated on both language and vision models across A100 and H100 GPUs, up to 128-GPU scale, with multi-dimensional parallelism and interconnects like InfiniBand and RoCE, TorchSim achieves over 90% accuracy in runtime prediction and 99% in memory estimation. It is open-sourced as an extension to PyTorch, with results demonstrated on TorchTitan.

1. Motivation and Introduction

Large AI models power a wide range of applications, but their training has become increasingly expensive and complex. Achieving state-of-the-art performance at this scale demands extreme computational investment. For example, Llama 3.1 used 405 billion parameters and 15 trillion tokens, consuming 30.84 million GPU hours across 16,000 H100s (Dubey et al., 2024), while Google's PaLM used 540 billion parameters and 0.8 trillion tokens, requiring 9.4 million TPU hours on 6,144 TPUv4 chips (Chowdhery et al., 2023). These efforts highlight not only the capabilities of modern models but also the prohibitive resource demands, making training cost a major bottleneck for scalable AI. Distributed training is essential for scaling, but no universally optimal strategy exists. For instance, Llama 3.1 employed 4D parallelism, 8-way Tensor, 16-way Context, 16-way Pipeline, and 8-way Fully Sharded Data Parallelism, while PaLM used 3D parallelism, 12-way Tensor, 256-way Fully Sharded Data, and 2-way Data Parallelism, each carefully tailored to model architecture and hardware constraints.

Scaling LLMs requires carefully combining parallelism strategies and system-level optimizations. This includes Data Parallelism (Li et al., 2020; Rajbhandari et al., 2020; Zhang et al., 2022a; Zhao et al., 2023), Tensor Parallelism(Narayanan et al., 2021; Wang et al., 2022; Korthikanti et al., 2023), Context Parallelism (Liu et al., 2023; Liu & Abbeel, 2024; NVIDIA, 2023; Fang & Zhao, 2024), and Pipeline Parallelism (Huang et al., 2019b; Narayanan et al., 2019; 2021; Tang et al., 2024b), often combined with techniques like activation recomputation (Chen et al., 2016; Korthikanti et al., 2023; He & Yu, 2023; Purandare et al., 2023), mixed precision (Micikevicius et al., 2018; 2022), and deep learning compilers (Bradbury et al., 2018; Yu et al., 2023; Li et al., 2024; Ansel et al., 2024b) to maximize efficiency. Moreover, identifying an effective training recipe is highly context-specific, requiring expert intuition and repeated experimentation across a large space of configurations involving parallelism dimensions, sharding strategies, memory trade-offs, and precision modes (Eisenman et al., 2022; Wang et al., 2023; Gupta et al., 2024; Maurya et al., 2024; Wan et al., 2024).

Even when frameworks support advanced optimizations, suboptimal configurations can be extremely costly. For instance, LLaMA 3.1 and PaLM consumed 30.84 million

[1]Harvard University, USA [2]Meta, USA. Correspondence to: Sanket Purandare <sanketpurandare@meta.com>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

RAGO: Systematic Performance Optimization for Retrieval-Augmented Generation Serving

Wenqi Jiang*
ETH Zurich
Zurich, Switzerland
wenqi.jiang@inf.ethz.ch

Suvinay Subramanian
Google
Mountain View, CA, USA
suvinay@google.com

Cat Graves
Google DeepMind
Mountain View, CA, USA
catgraves@google.com

Gustavo Alonso
ETH Zurich
Zurich, Switzerland
alonso@inf.ethz.ch

Amir Yazdanbakhsh†
Google DeepMind
Mountain View, CA, USA
ayazdan@google.com

Vidushi Dadu†
Google
Sunnyvale, USA
vidushid@google.com

Abstract

Retrieval-augmented generation (RAG) is emerging as a popular approach for reliable LLM serving. However, efficient RAG serving remains an open challenge due to the rapid emergence of many RAG variants and the substantial differences in workload characteristics across them. This paper makes three fundamental contributions to advancing RAG serving. First, we introduce **RAGSchema**, a structured abstraction that captures the wide range of RAG algorithms, serving as a foundation for performance optimization. Second, we analyze several representative RAG workloads with distinct RAGSchema, revealing significant performance variability across these workloads. Third, to address this variability and meet diverse performance requirements, we propose **RAGO** (**R**etrieval-**A**ugmented **G**eneration **O**ptimizer), a system optimization framework for efficient RAG serving. RAGO achieves up to a 2× increase in QPS per chip and a 55% reduction in time-to-first-token latency compared to RAG systems built on LLM-system extensions.

CCS Concepts

• **Information systems** → **Information retrieval**; • **Computer systems organization** → **Architectures**; • **Computing methodologies** → **Natural language processing**; • **Hardware** → **Emerging simulation**.

Keywords

Retrieval-Augmented Generation, Computer System, Computer Architecture, Large Language Model, Performance Optimization

ACM Reference Format:
Wenqi Jiang, Suvinay Subramanian, Cat Graves, Gustavo Alonso, Amir Yazdanbakhsh, and Vidushi Dadu. 2025. RAGO: Systematic Performance Optimization for Retrieval-Augmented Generation Serving. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3695053.3731093

*Work done while at Google.
†Equal advising.

Artifact: https://github.com/google/rago.

ISCA '25, Tokyo, Japan

ACM ISBN 979-8-4007-1261-6/25/06
https://doi.org/10.1145/3695053.3731093

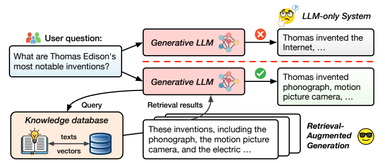


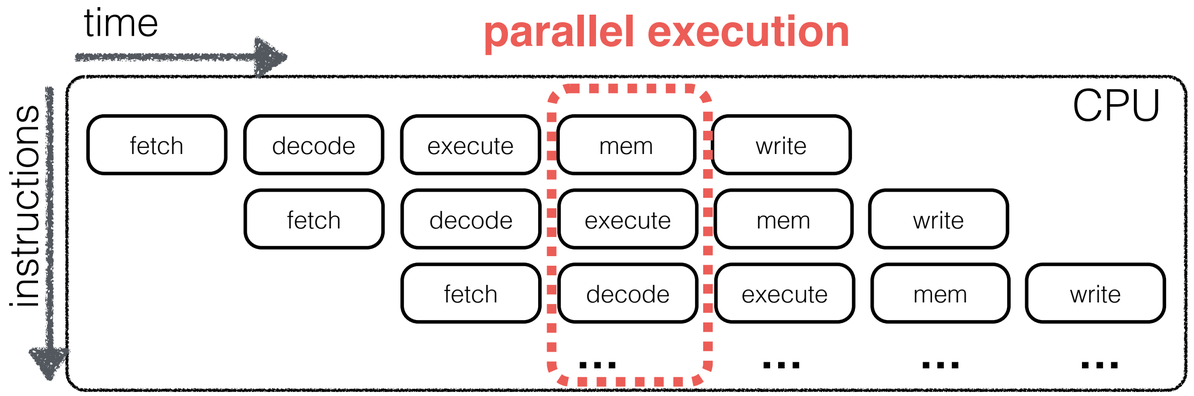
Figure 1: LLM-only system (top) versus RAG (bottom).

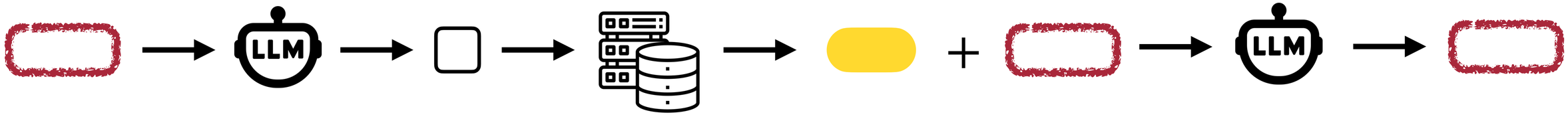
1 Introduction

The rapid adoption of Large Language Models (LLMs) across diverse applications — such as question answering [23, 26, 29], code generation [73, 76, 90], and scientific discoveries [20, 67, 107] — showcases their profound impact on automating knowledge-based tasks. Despite these capabilities, LLM systems, when deployed in isolation (aka LLM-only systems), face substantial challenges, such as data staleness [63, 71], a propensity to hallucinate (generating factually incorrect or nonsensical information), and limited, often rigid model knowledge [46, 71, 74]. These challenges hinder the reliability and adaptability of LLM-only systems, especially in applications that demand high factual accuracy [31, 42, 57, 109].

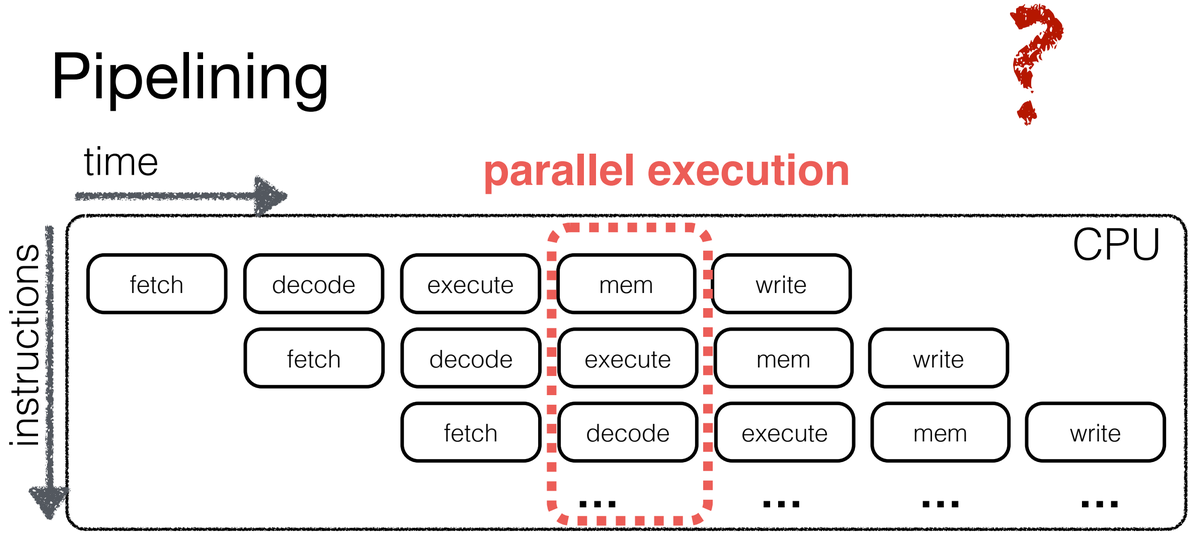
Retrieval-Augmented Generation (RAG) has emerged as one powerful solution to address the common pitfalls of LLM-only systems for knowledge-intensive tasks [22, 63, 70, 99, 112]. By retrieving information from external databases and appending it to prompts (Figure 1), RAG enhances the credibility, timeliness, and contextually rich nature of LLM-generated responses. Leveraging the generative prowess of LLMs alongside external knowledge sources, RAG not only achieves comparable quality with smaller models [22, 99, 112] but also simplifies the process of updating knowledge, mitigating the extent of additional model training, which is often prohibitively expensive [63, 70]. These advantages have established RAG as the industry standard for knowledge-intensive applications, with notable examples including Google's REALM [39] and RETRO [22], Meta's MARGE [70], Microsoft's GraphRAG [30], and NVIDIA's InstructRETRO [112]. As companies race to integrate RAG systems into their production pipelines [28, 84, 97], optimizing their performance has become increasingly critical.

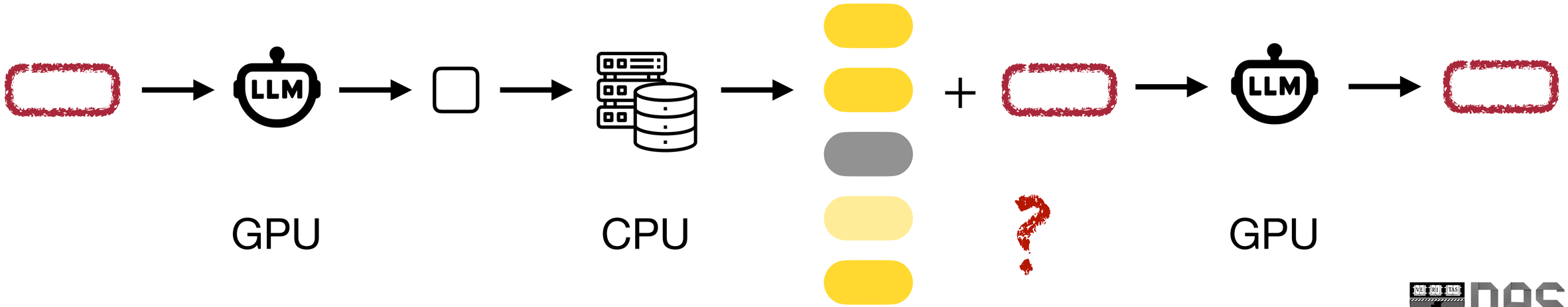
Pipelining

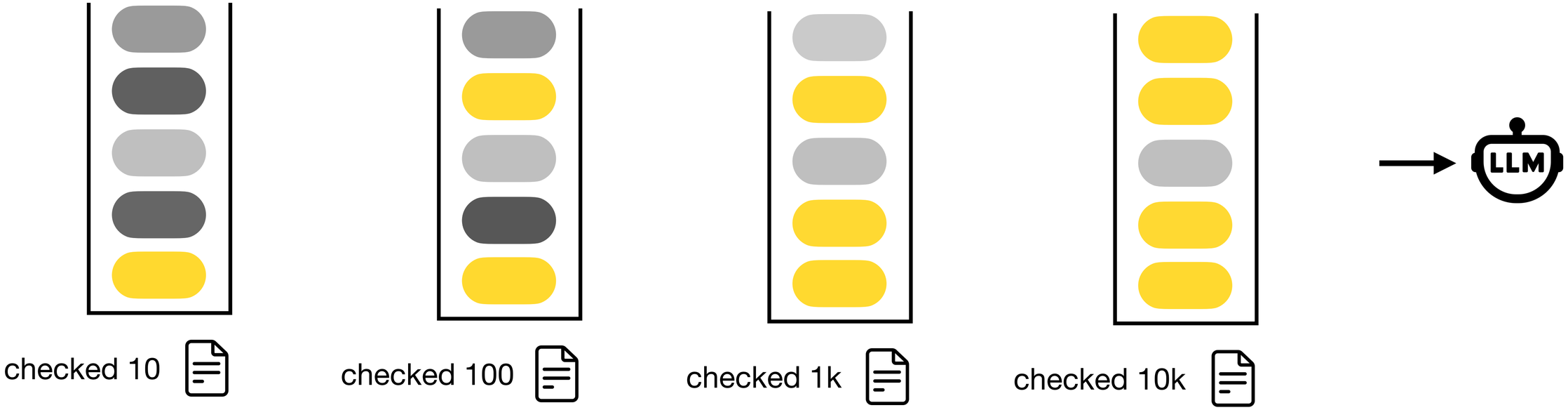


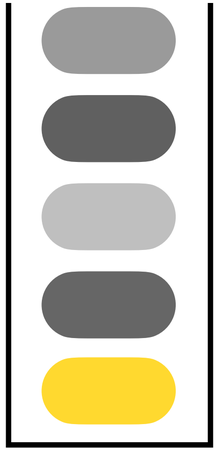


GPU CPU GPU

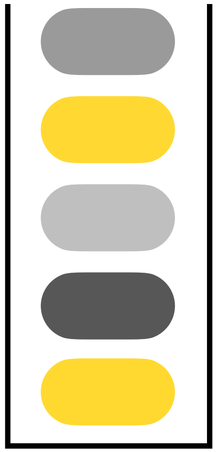




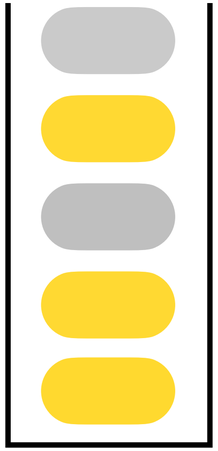




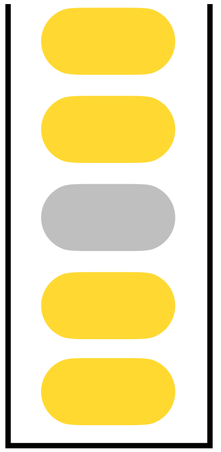
checked 10



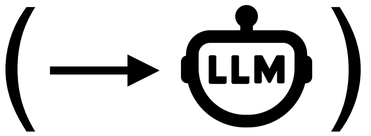
checked 100



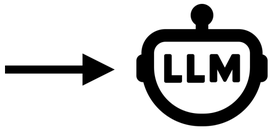
checked 1k

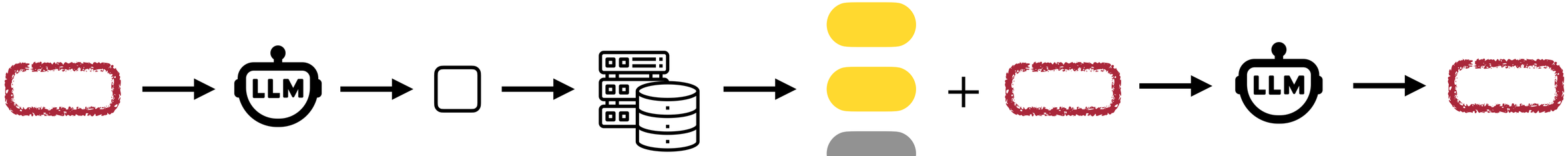


checked 10k



restart (partially)
only if
documents change



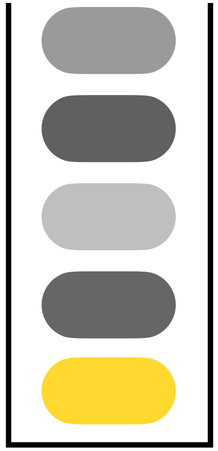


GPU

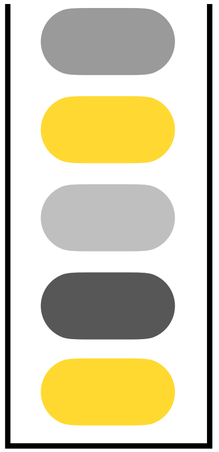
CPU

GPU

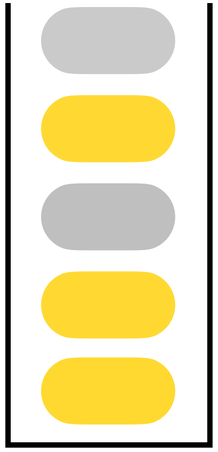
DASlab
@ Harvard SEAS



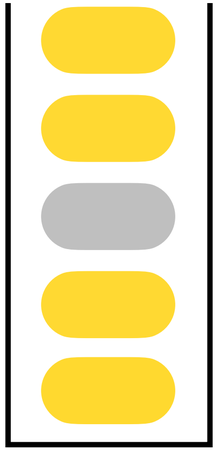
checked 10



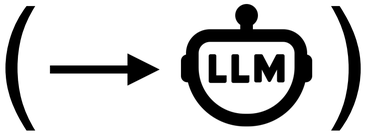
checked 100



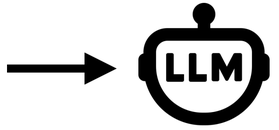
checked 1k

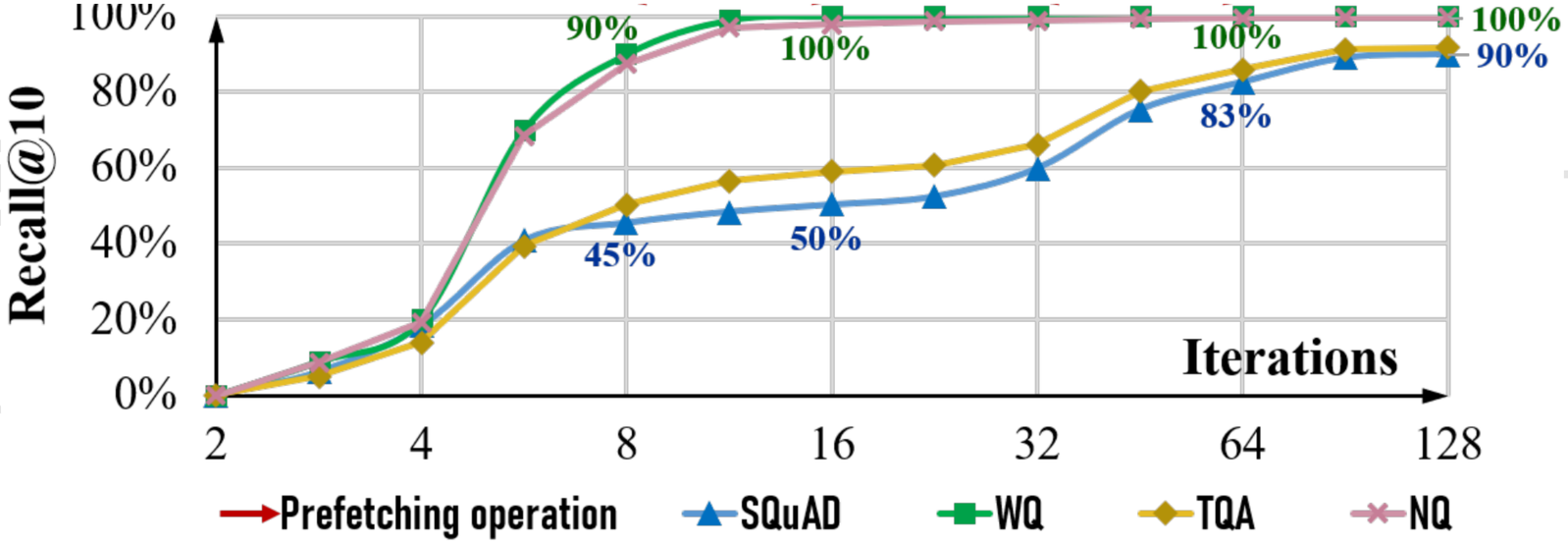


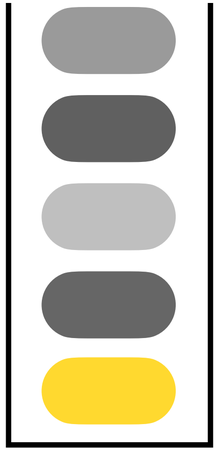
checked 10k



restart (partially)
only if
documents change







checked 10

checked 100

AquaPipe: A Quality-Aware Pipeline for Knowledge Retrieval and Large Language Models

RUNJIE YU, Huazhong University of Science and Technology, China
WEIZHOU HUANG, Huazhong University of Science and Technology, China
SHUHAN BAI, Huazhong University of Science and Technology, China
JIAN ZHOU*, Huazhong University of Science and Technology, China
FEI WU*, Huazhong University of Science and Technology, China

The knowledge retrieval methods such as Approximate Nearest Neighbor Search (ANNS) significantly enhance the generation quality of Large Language Models (LLMs) by introducing external knowledge, and this method is called Retrieval-augmented generation (RAG). However, due to the rapid growth of data size, ANNS tends to store large-scale data on disk, which greatly increases the response time of RAG systems. This paper presents AquaPipe, which pipelines the execution of disk-based ANNS and the LLM *prefill* phase in an RAG system, effectively overlapping the latency of knowledge retrieval and model inference to enhance the overall performance, while guaranteeing data quality. First, ANNS's recall-aware prefetching strategy enables the early return of partial text with acceptable accuracy so the prefill phase can launch before getting the full results. Then, we adaptively choose the remove-after-prefill or re-prefill strategies based on the LLM cost model to effectively correct disturbed pipelines caused by wrong early returns. Finally, the pipelined prefill dynamically changes the granularity of chunk size to balance the overlap efficiency and GPU efficiency, adjusting to ANNS tasks that converge at different speeds. Our experiments have demonstrated the effectiveness of AquaPipe. It successfully masks the latency of disk-based ANNS by 56% to 99%, resulting in a 1.3× to 2.6× reduction of the response time of the RAG, while the extra recall loss caused by prefetching is limited to approximately 1%.

CCS Concepts: • **Information systems** → **Nearest-neighbor search**; **Search engine indexing**.

Additional Key Words and Phrases: Approximate Nearest Neighbor Search, Retrieval-augmented Generation, Large Language Models

ACM Reference Format:
Runjie Yu, Weizhou Huang, Shuhan Bai, Jian Zhou, and Fei Wu. 2025. AquaPipe: A Quality-Aware Pipeline for Knowledge Retrieval and Large Language Models. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 11 (January 2025), 26 pages. https://doi.org/10.1145/3709661

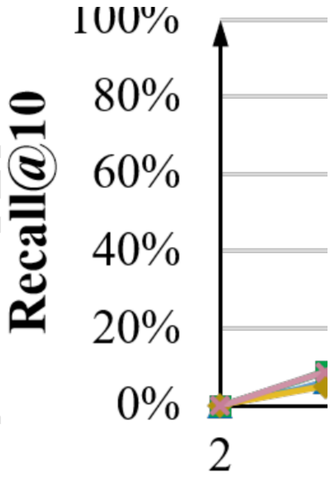
1 Introduction

Large Language Models (LLMs) have demonstrated impressive capabilities in understanding and generating human languages. However, due to the limitation of their training datasets, they face challenges such as hallucinations [70], outdated knowledge, and untraceable reasoning processes. To

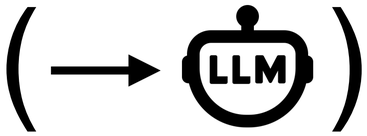
*Jian Zhou and Fei Wu are co-corresponding authors.

Authors' Contact Information: Runjie Yu, d202381500@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China; Weizhou Huang, huangweizhou@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China; Shuhan Bai, shuhanbai0329@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China; Jian Zhou, jianzhou@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China; Fei Wu, wufei@mail.hust.edu.cn, Huazhong University of Science and Technology, Wuhan, Hubei, China.

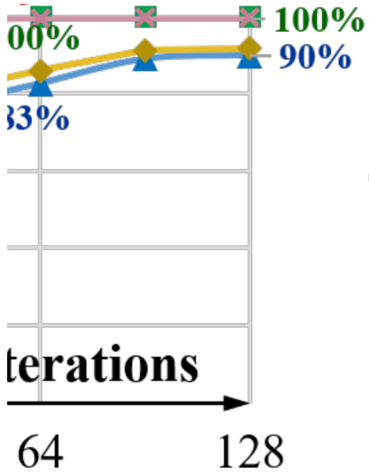
ACM 2836-6573/2025/1-ART11
https://doi.org/10.1145/3709661

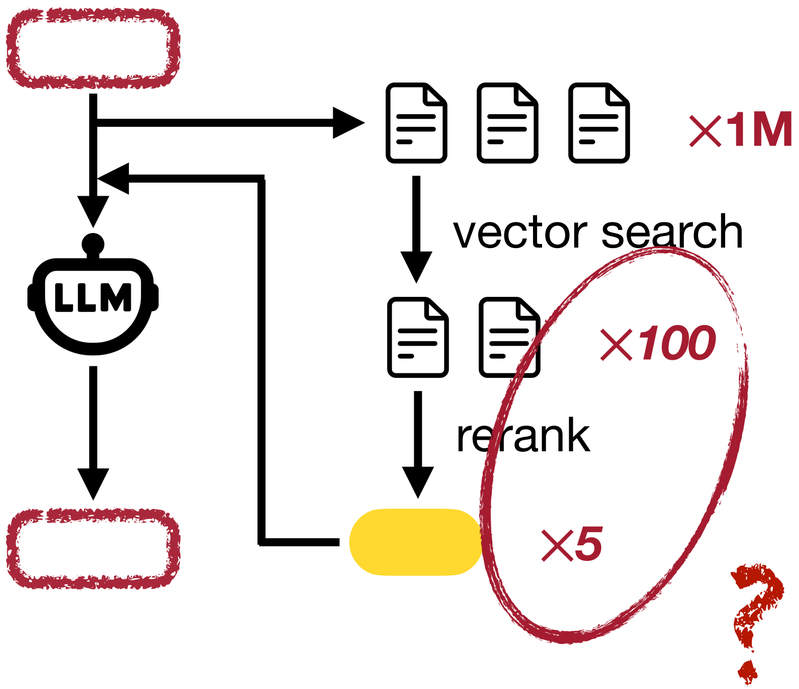


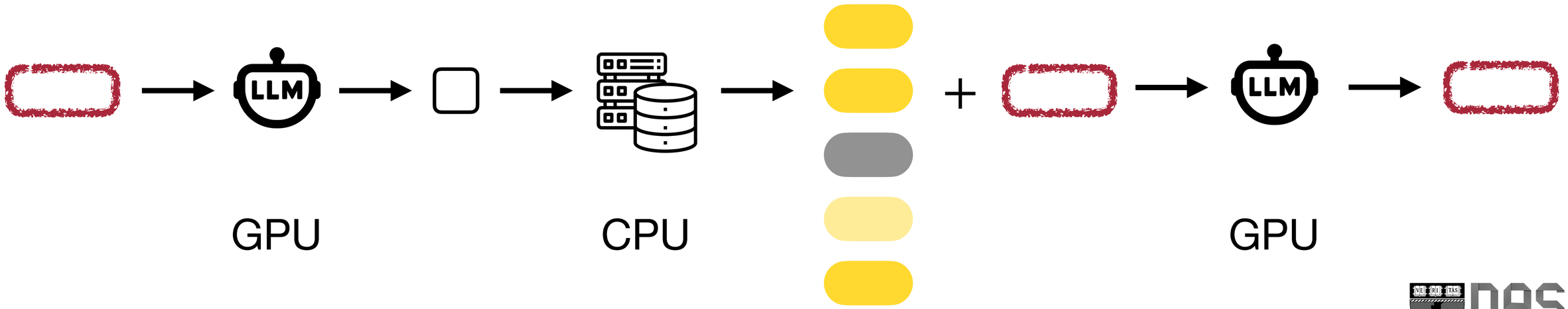
0k

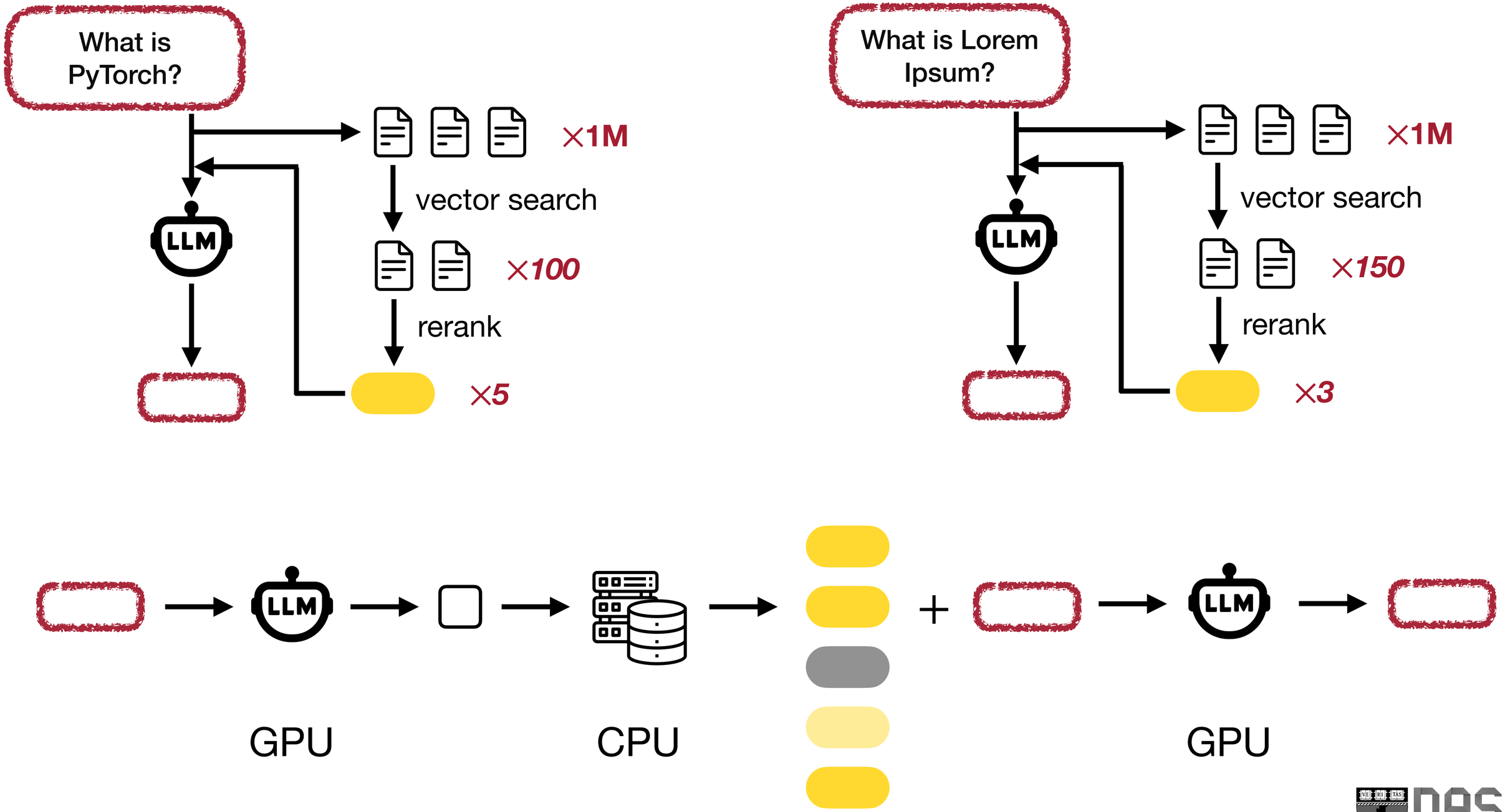


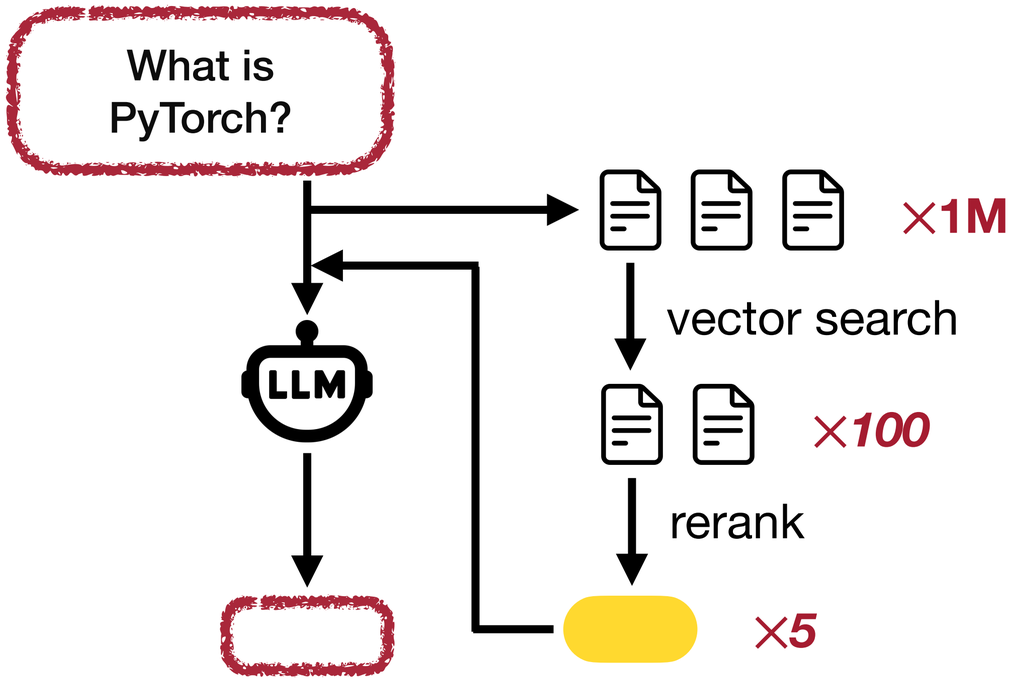
restart (partially)
only if
documents change

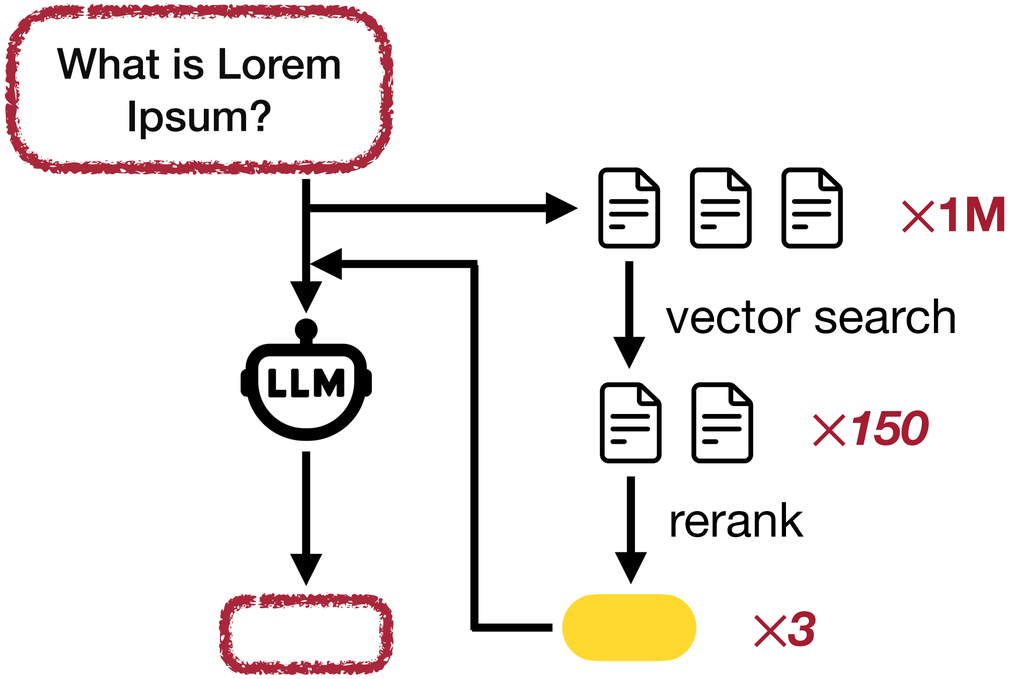




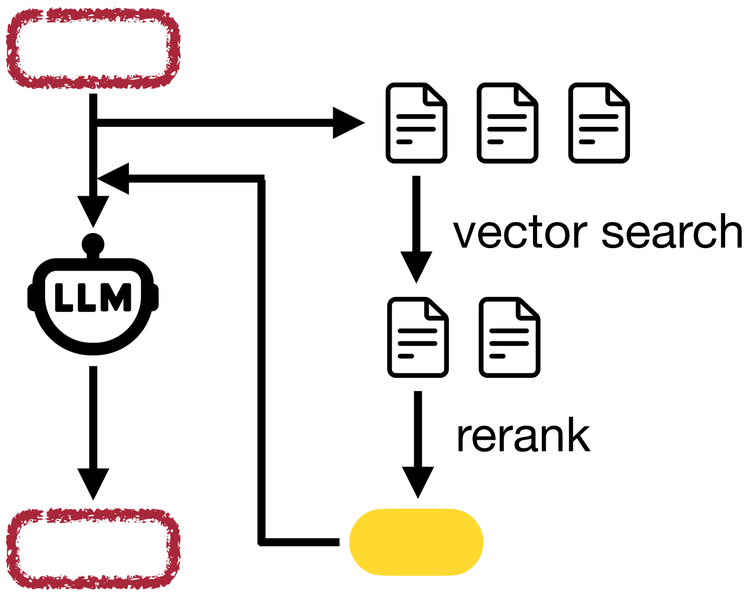








This is an on-going work in the lab, and you are welcome to join us!



Resource allocation:
*adapt to the **hardware***

Auto-configuration:
*adapt to the **application***

Pipelining, …:
*adapt to the **performance***

Self-designing RAG systems!