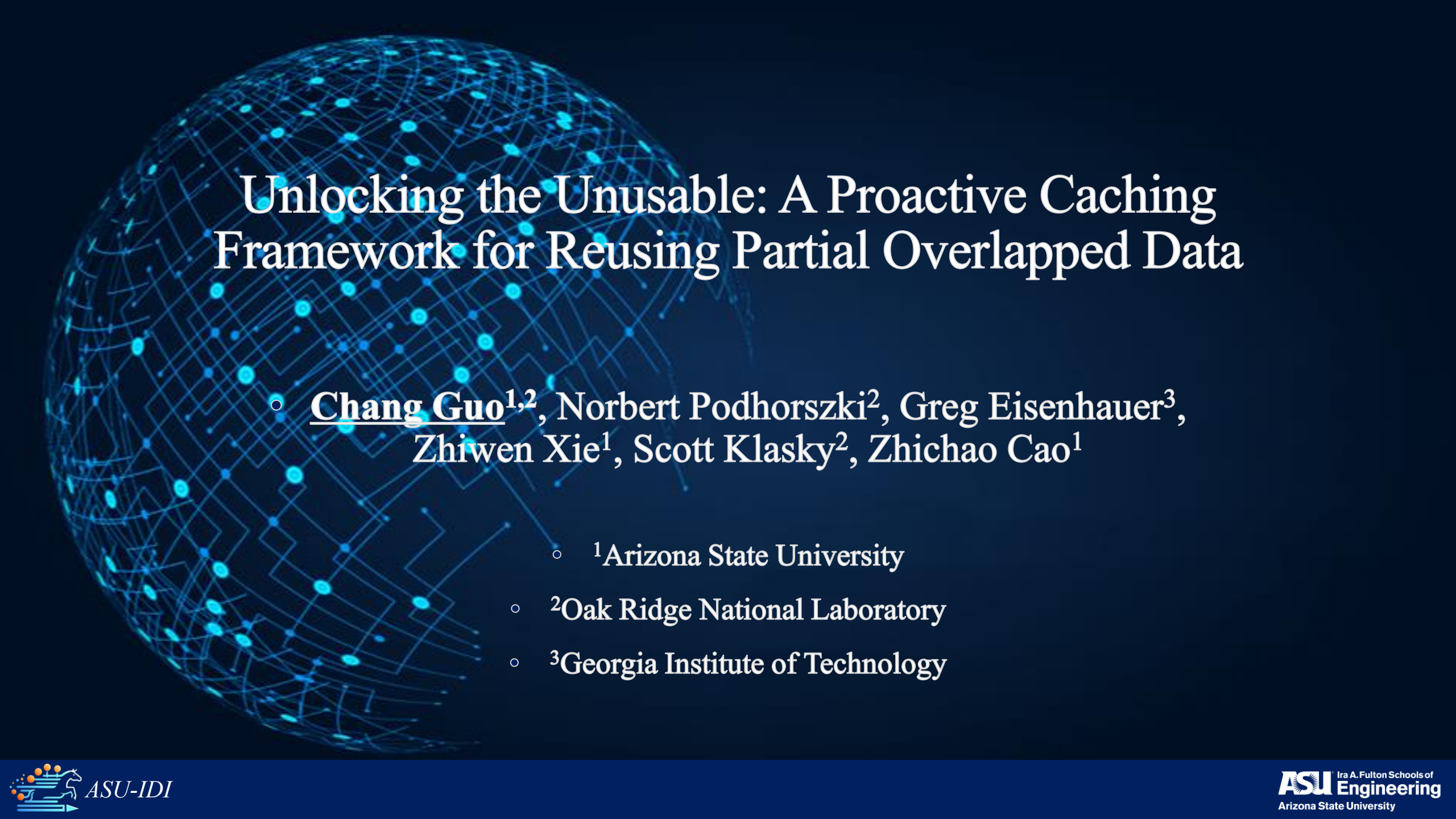


Unlocking the Unusable: A Proactive Caching Framework for Reusing Partial Overlapped Data

- **Chang Guo**[1,2], Norbert Podhorszki[2], Greg Eisenhauer[3], Zhiwen Xie[1], Scott Klasky[2], Zhichao Cao[1]

- [1]Arizona State University
- [2]Oak Ridge National Laboratory
- [3]Georgia Institute of Technology

Background & Motivation

- Traditional cache engines (e.g., Redis, CacheLib, Memcached) rely on exact key matches:

Insert: **Key123** → Value123

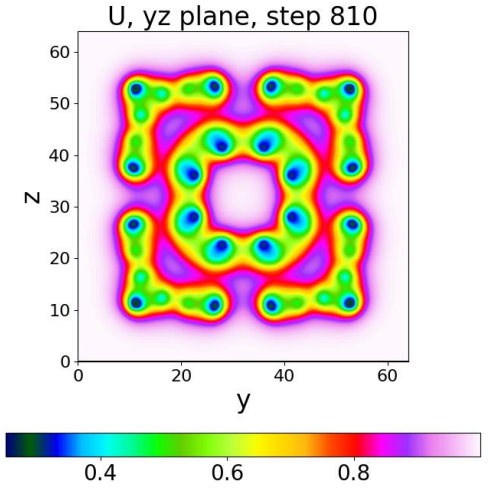
Lookup: **Key123** ✔

Lookup: **Key124** ✘

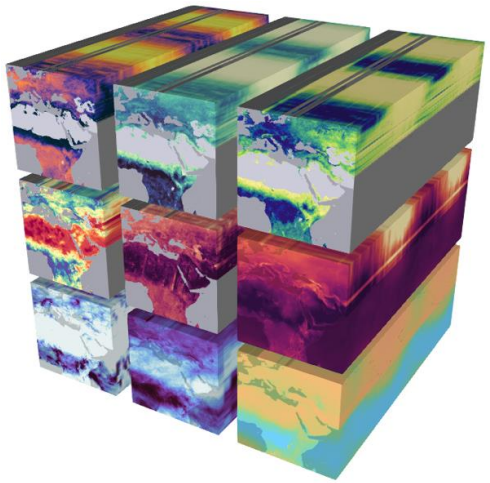
(miss, even if the data it refers to is largely **overlapped**)

Background & Motivation

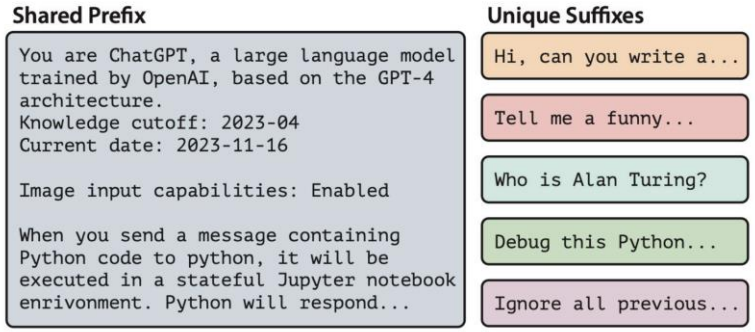
- Many emerging applications exhibit **complex and irregular** data layouts:



HPC: Multi-dimensional Fields



Earth Observation: Sensors



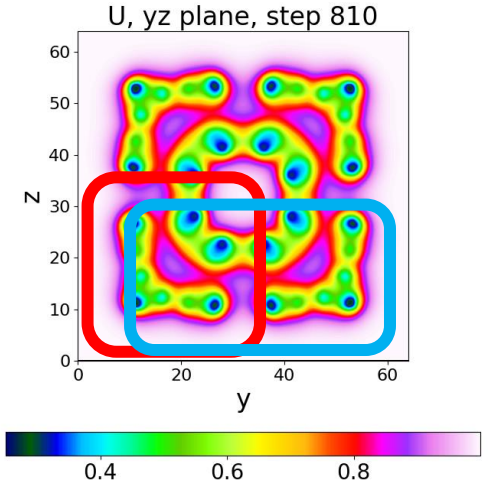
LLM: Long Context

Background & Motivation

- And the requested data from different queries are **partially overlapped**:

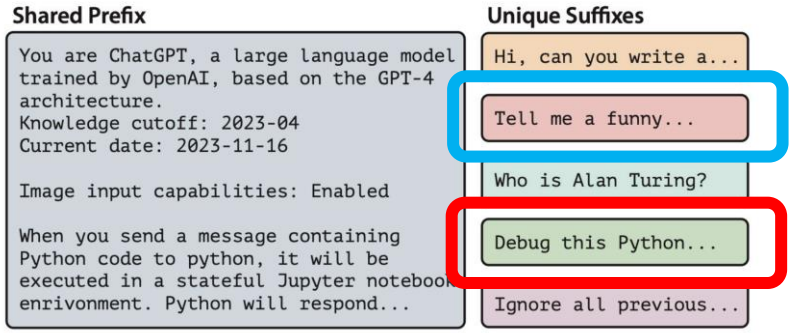
Previous Request

New Request



HPC: Multi-dimensional Fields

Earth Observation: Sensors



LLM: Long Context

Limitations of Traditional Caches: No Reuse of Partially Overlapped Data!

Background & Motivation

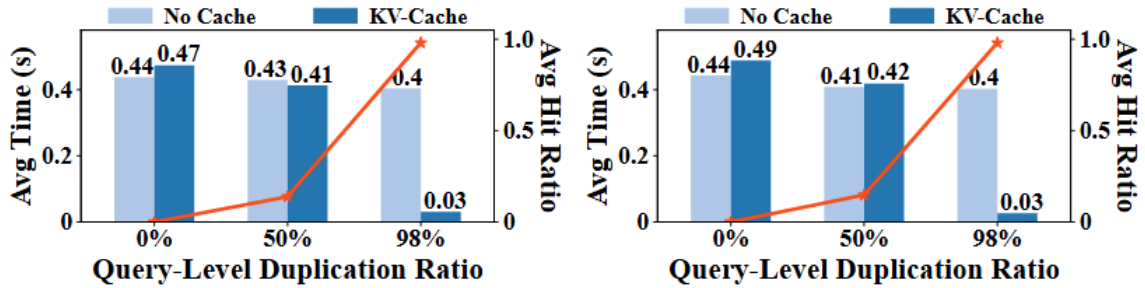
- In many applications, exact query repetition is rare [23, 41], ❌
- Content-level overlap across queries is more common [8, 22]. ✅
- However,
  1. No mechanism to convert metadata into simple **unique** cache keys while **identifying** partial overlaps.
  2. Typical **Hash-based indexing** only supports exact matches, resulting in low hit rates and inefficiency.
  3. Directly caching overlapped data causes duplication across items, wasting cache capacity.

Background & Motivation

We conduct a simple experiment with cache size: ~10% of the total dataset

Query-Level Duplication: exact duplicate queries in the dataset

Content-Level Overlap: content similarity between different queries, even if they are not identical



(a) Content-Level Overlap Ratio: 0% (b) Content-Level Overlap Ratio: 50%

Figure 1: Performance with Varying Overlap Ratios.

Similar Performance with Different Content-Level Overlaps →

Traditional Cache Engines Fail to **Reuse Partially Overlapped Data!**

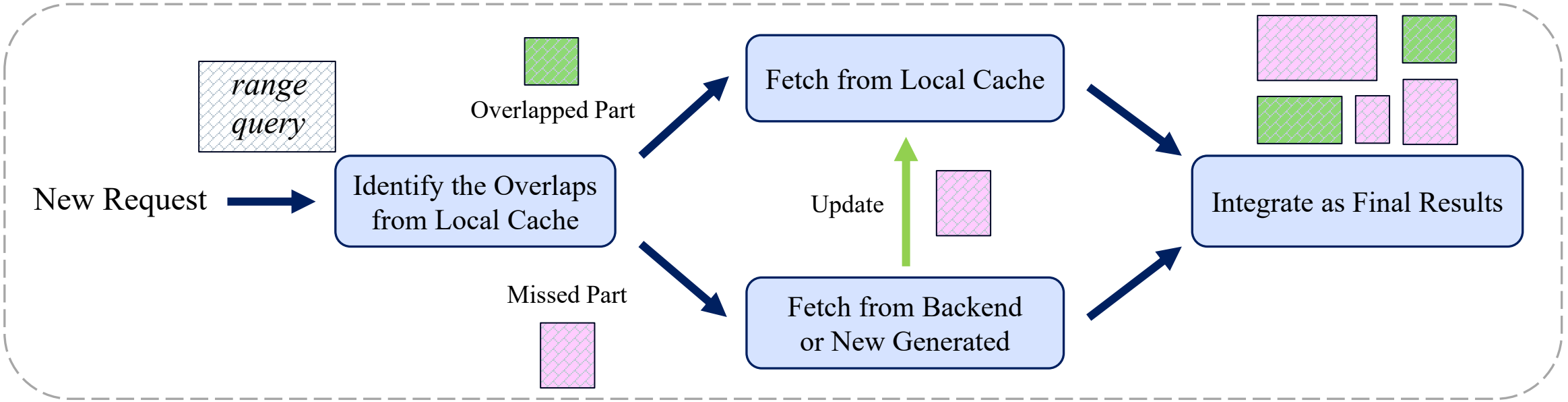
Research Objective and Challenges

Design a cache framework that can actively **reuse partially overlapping content** from cached items to improve overall application performance.

Research Objective and Challenges

Design a cache framework that can actively **reuse partially overlapping content** from cached items to improve overall application performance.

The workflow of reusing partial overlapping content:

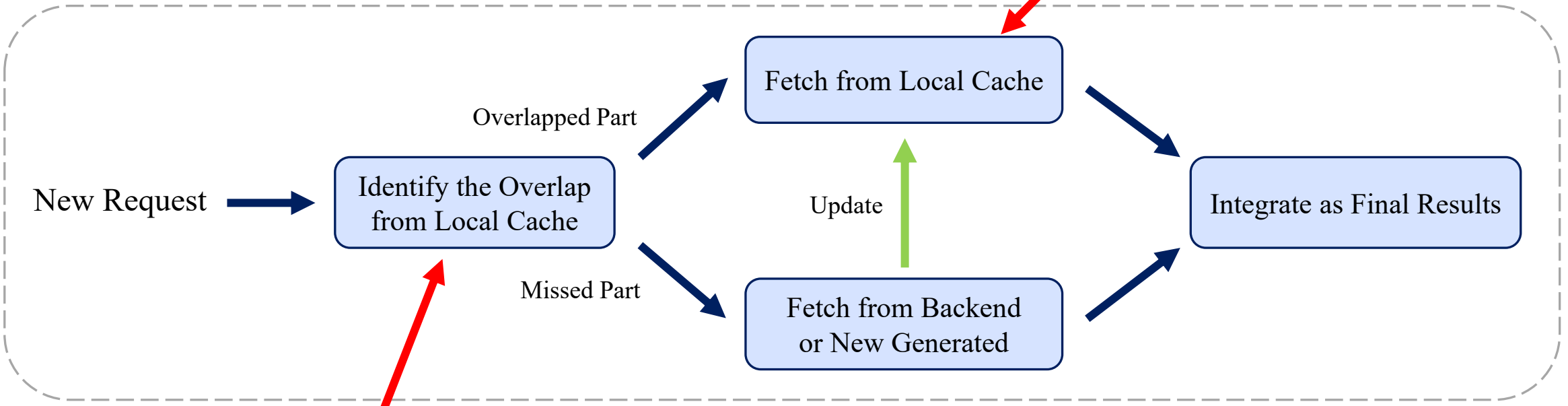


Research Objective and Challenges

However, this workflow comes with the following challenges:

① [Generality] How can we support diverse data structures?
E.g., various high-dimensional or nested data.

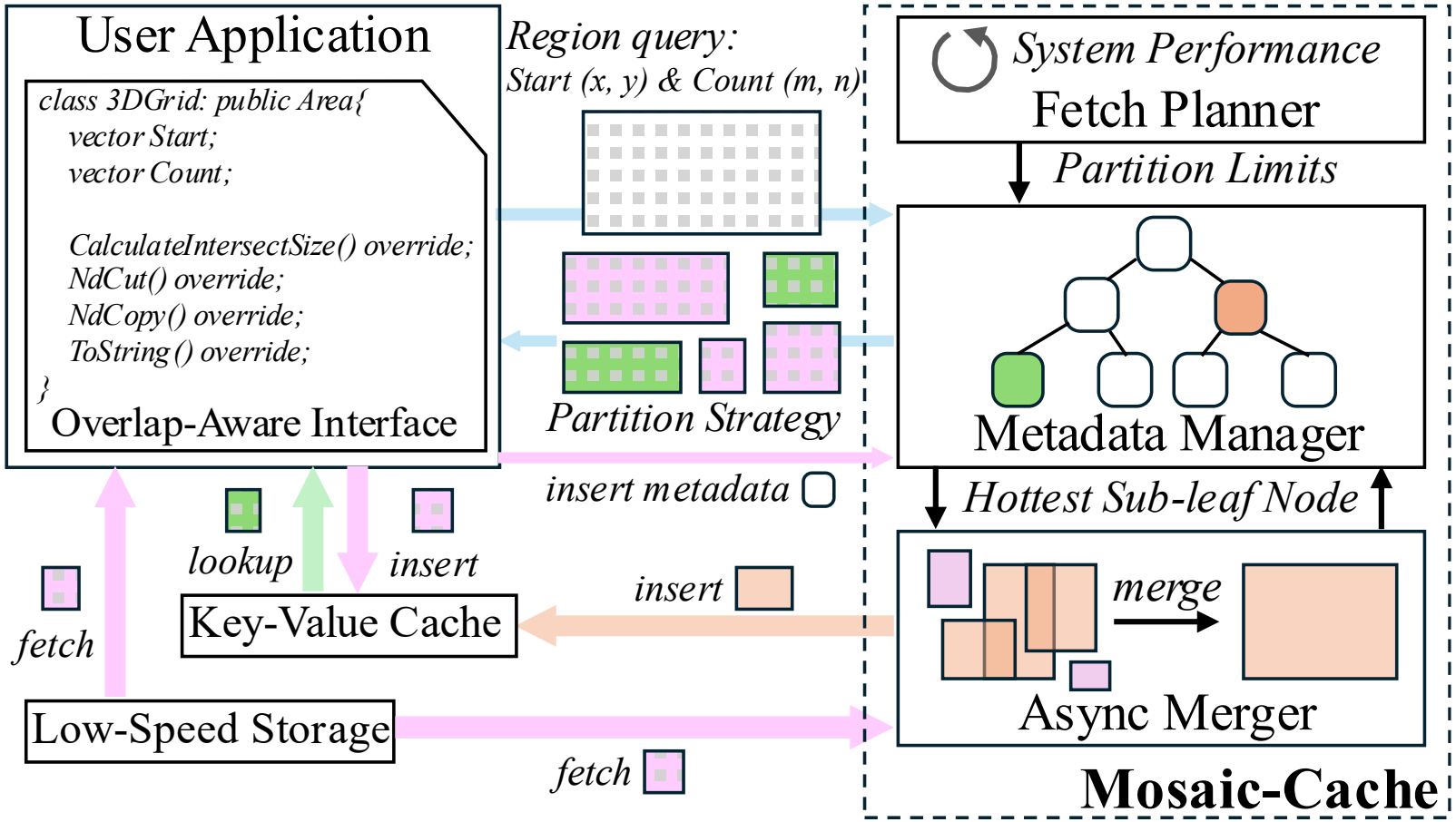
④ [Duplication and Fragment]
How should local cache be managed?



② [Efficiency] How can we efficiently parse complex overlaps?
(*Traditional cache **keys** struggle to identify partial overlaps.*)

③ [Trade-off] When should we use local cache?
(*Balancing performance benefits vs. overheads.*)

Mosaic-Cache Design Overview



① [Generality]

② [Efficiency]

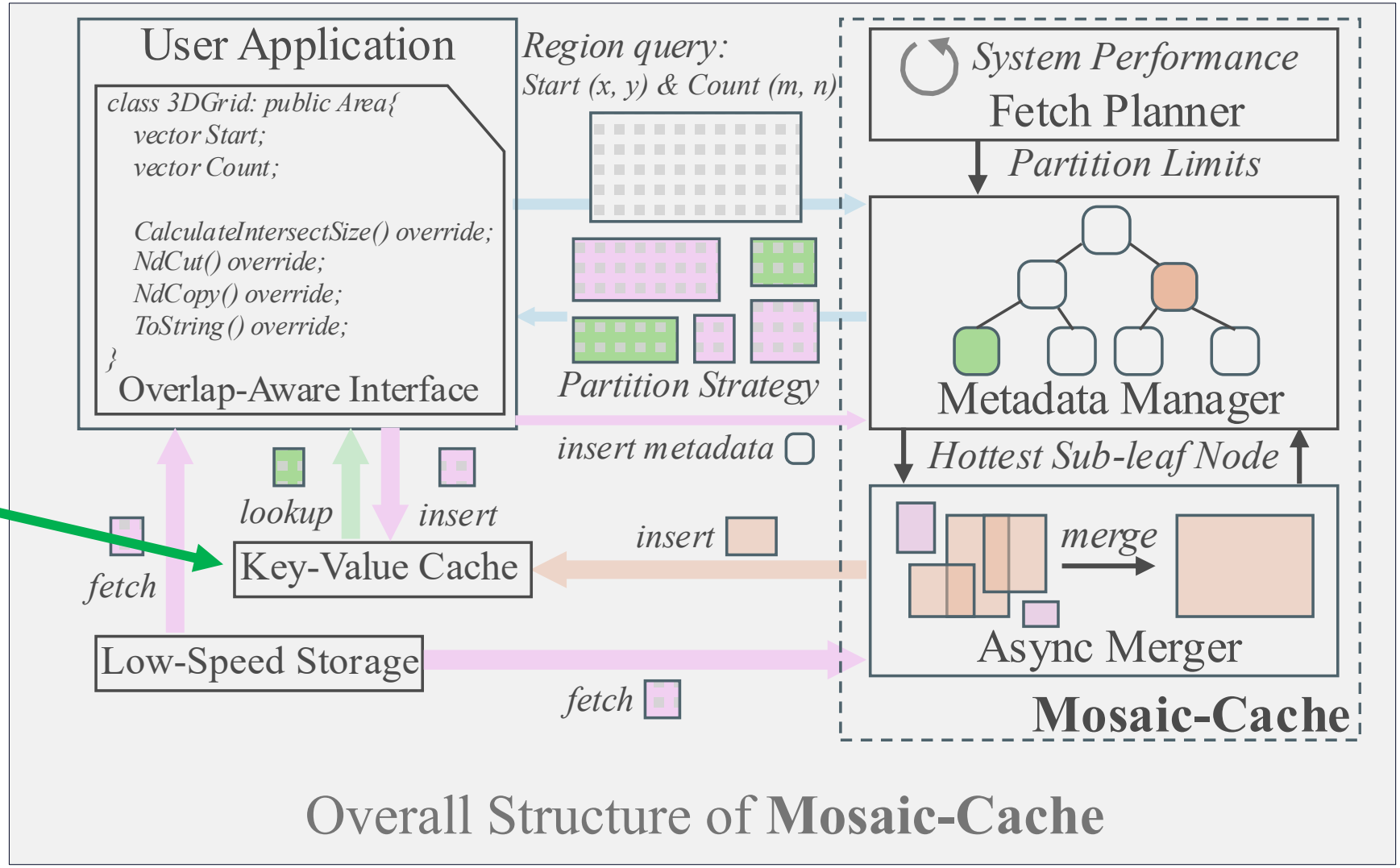
③ [Trade-off]

④ [Duplication and Fragment]

Overall Structure of **Mosaic-Cache**

Mosaic-Cache Design Overview

Compatible with traditional caches to avoid reinventing the wheel

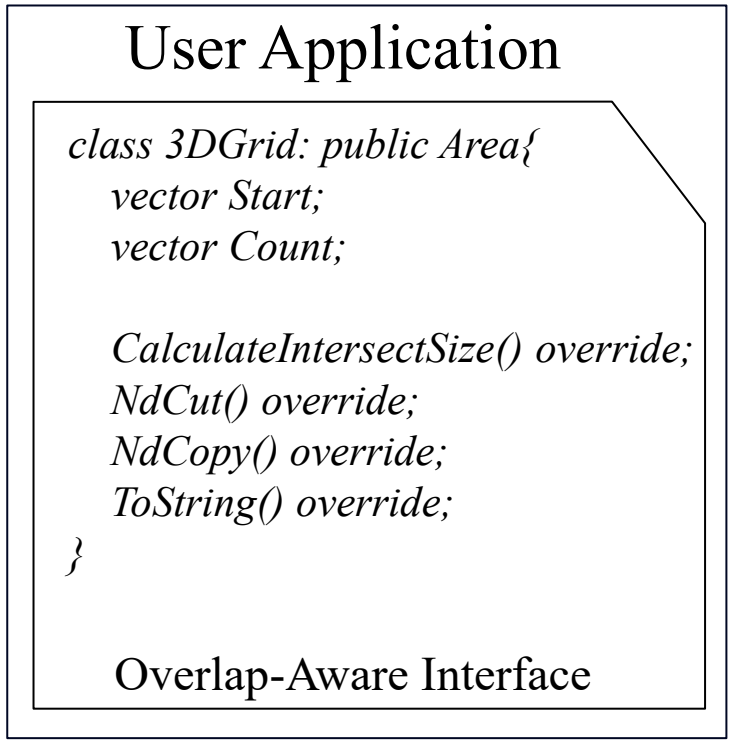


Mosaic-Cache Design

1. Overlap-Aware Interface ① [Generality]

Mosaic-Cache pre-define user-customized interfaces to allow user to design:

- Complex data structure
- How to calculate the intersect size
- How to extract the overlapped area from previous requests
- How to integrate the pieces together
- Convert complex metadata into unique identifiers

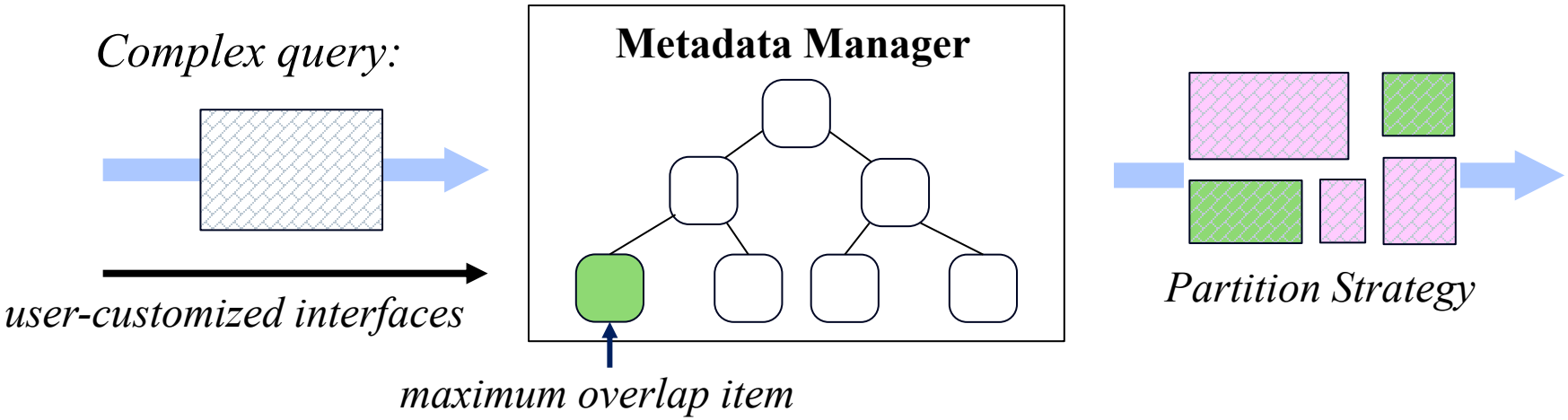


Mosaic-Cache Design

2. Metadata Manager ② [Efficiency]

Mosaic-Cache provides an in-memory manager to handle complex metadata, enabling:

- Efficient indexing without frequent key serialization/deserialization
- Recursively identify maximum overlap items to generate the partition strategy
- Customizable parameters (e.g., *visit_num*) to track access frequency and data hotness



Mosaic-Cache Design

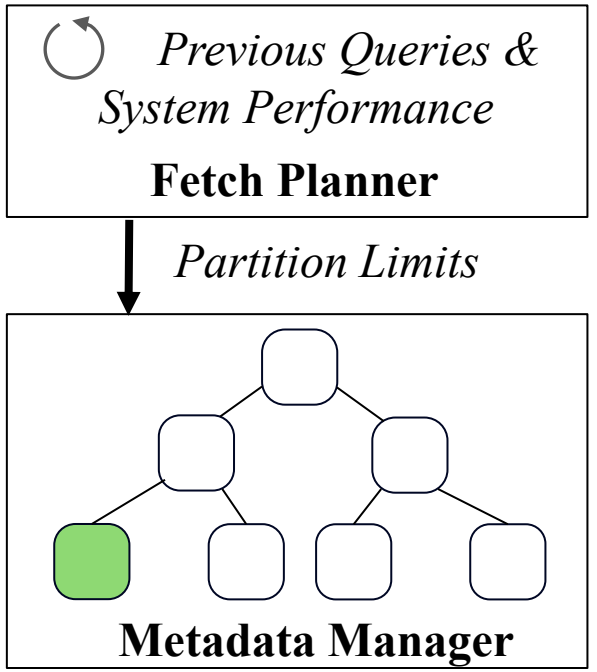
3. Fetch Planner ③ [Trade-off]

Identifying, partitioning, and integrating overlapping introduce extra overheads.

- Sometimes directly fetching data from the backend is more efficient.

Mosaic-Cache introduces a **Fetch Planner** to

- Continuously collects previous queries and system performance
- Determine how much of the local cache to reuse

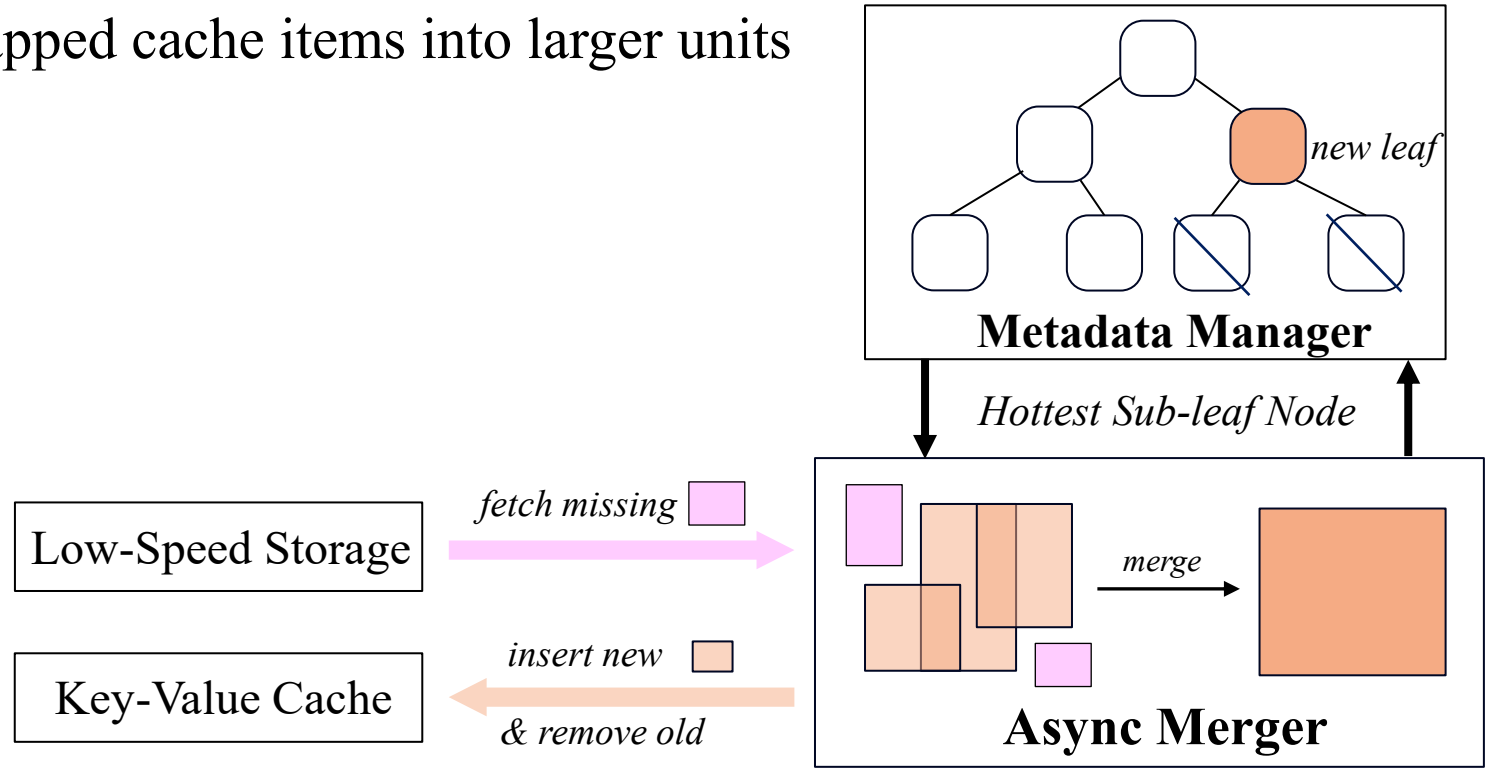


Mosaic-Cache Design

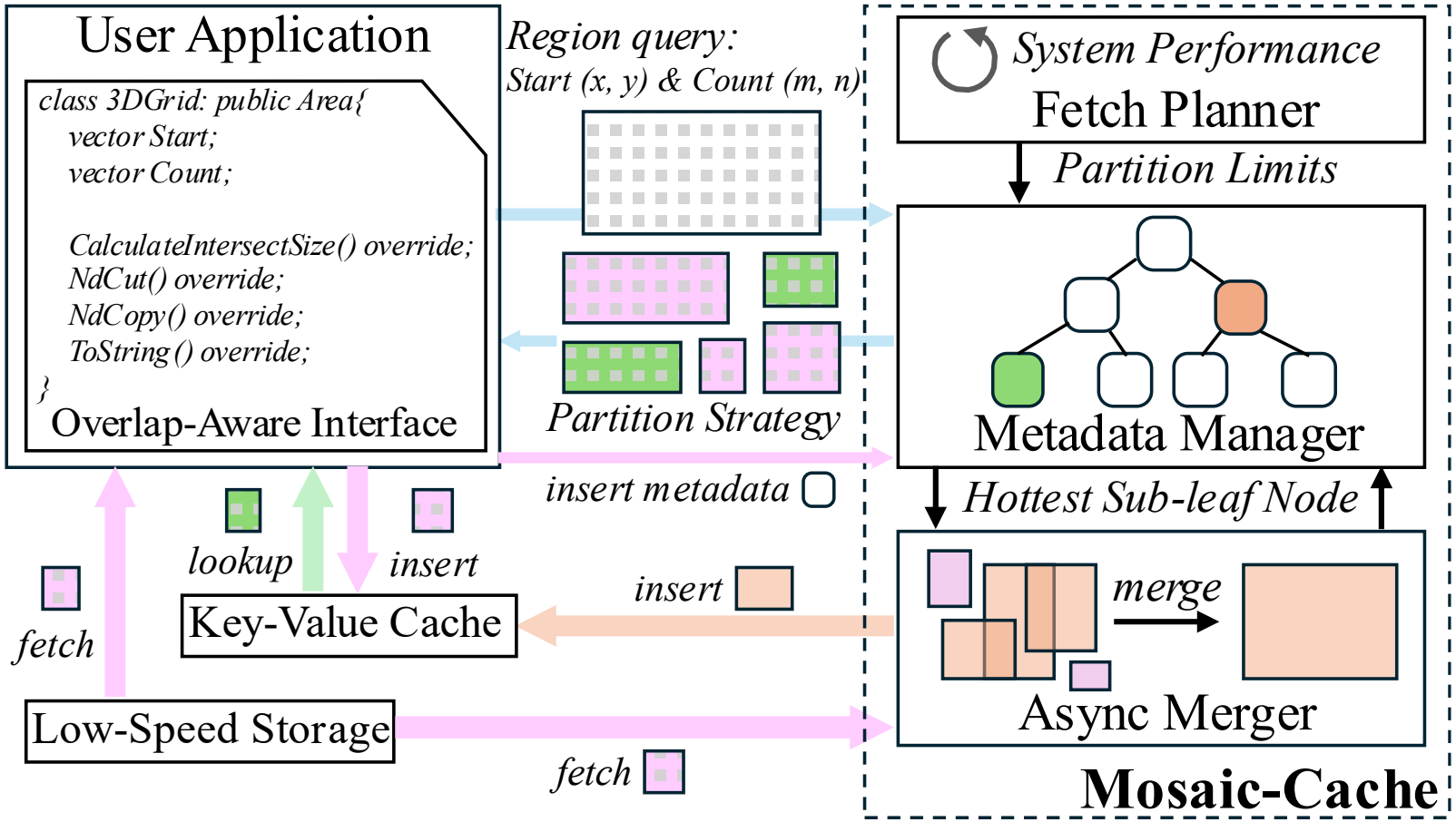
4. Async Merger ④ [Duplication and Fragment]

Mosaic-Cache introduces an **Async Merger** to

- Merge hot and partially overlapped cache items into larger units



Mosaic-Cache Design Overview



① [Generality]

② [Efficiency]

③ [Trade-off]

④ [Duplication and Fragment]

Overall Structure of **Mosaic-Cache**

Evaluation

Scenario: 3D Combustion Grid Data Analysis (ORNL)

Data Size: ~1.5 TB

Application: ADIOS2 (Adaptable Input/Output System v2)

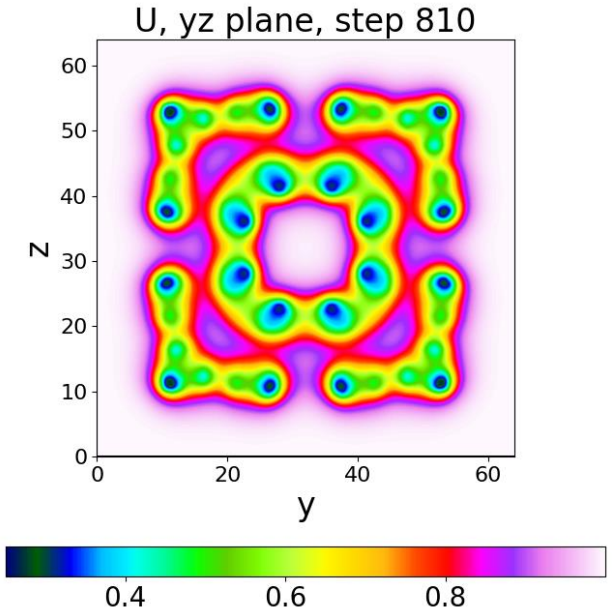
Query Workload: 1,000 randomly ordered range queries per test

Query Size: ~10 MB each → ~10 GB total per test

Cache Size: 1GB, 10% of the total

Baseline:

- No Cache: All from Backend (ORNL Remote FS)
- KV-Cache: Traditional Cache Engine
- Meta-Match: Traditional Cache Engine (Parse all cache keys from strings)



HPC: Multi-dimensional Fields

Evaluation

1. Impact of Overlap

Setting: No Duplicate Queries; Varying Content Overlap (0%–98%)

No Cache & KV-Cache:

- Stable, high latency (no overlap benefit)

Meta-Match:

- Latency first drops with overlap
- Rises again at high overlap due to key parsing

MOSAIC:

- Latency keeps decreasing
- Obvious performance gains when overlap > 80%

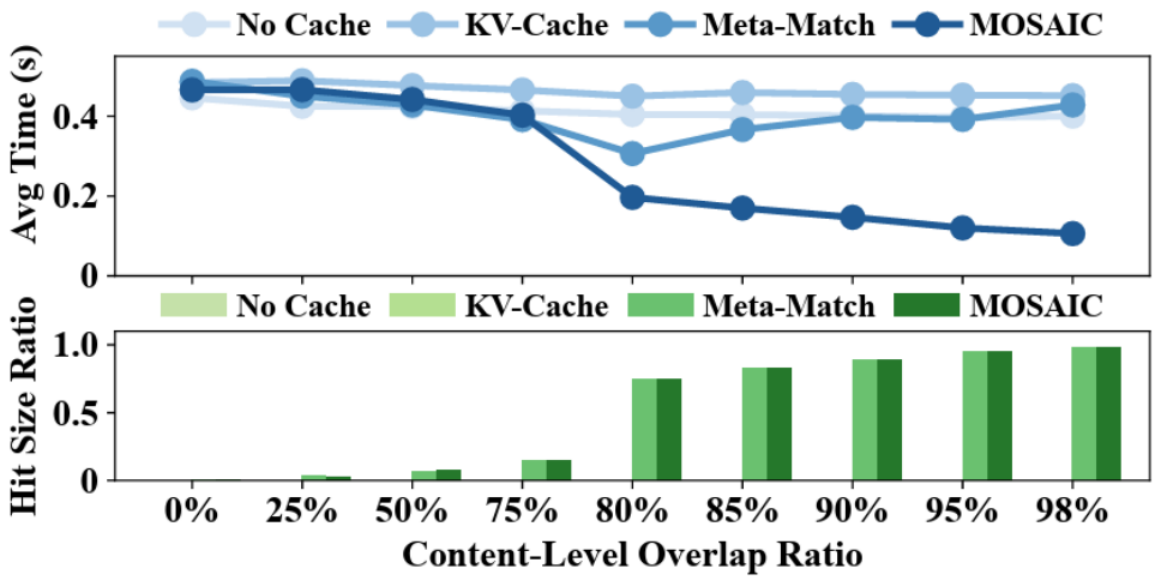


Figure 3: Impact of Overlap on Query Performance.

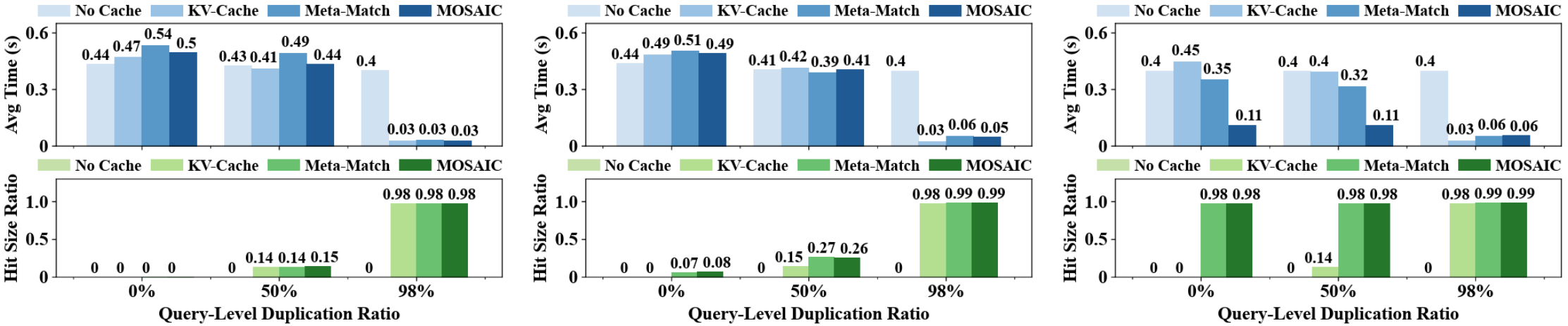
Evaluation

2. Overall Performance

No Overlap (0%): All caches show overhead, Mosaic-Cache ≈ others

Small Overlap (50%): Mosaic-Cache & Meta-Match outperform others via partial reuse

High Overlap (98%): Mosaic-Cache shows best performance (No duplicate queries.)



(a) Content-Level Overlap Ratio: 0%

(b) Content-Level Overlap Ratio: 50%

(c) Content-Level Overlap Ratio: 98%

Figure 4: Overall Performance Comparison Across Query-Level Duplication and Content-Level Overlap Ratios.

Evaluation

3. Impact of Cache Size

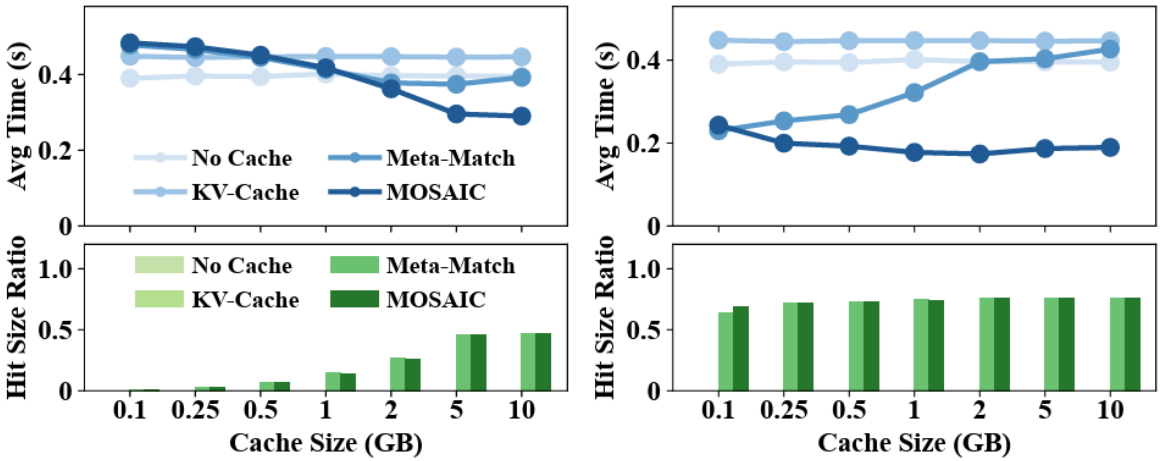
Varying Cache Size to explain MOSAIC's sharp performance jump from 75% → 80% overlap

At 75% overlap:

- Needs ≥5 GB cache for optimal performance
- Smaller caches → early eviction & lower hit ratio

At 80% overlap:

- 1 GB cache is sufficient



(a) Content-Level Overlap Ratio: 75% (b) Content-Level Overlap Ratio: 80%

Figure 5: Impact of Cache Size on Query Performance.

Evaluation

4. Impact of Backend Access Latency

Vary storage latency: **2 ms (local same shelf) → 70 ms (Remote FS),** 0% exact match, 80% overlap

MOSAIC-Cache

- Consistently outperforms baselines across all latency setups
- At low latency, Mosaic favors **direct access** to reduce overhead
  - lower hit ratio, but better performance

Meta-Match

- High hit ratio, but suffers from reuse overhead

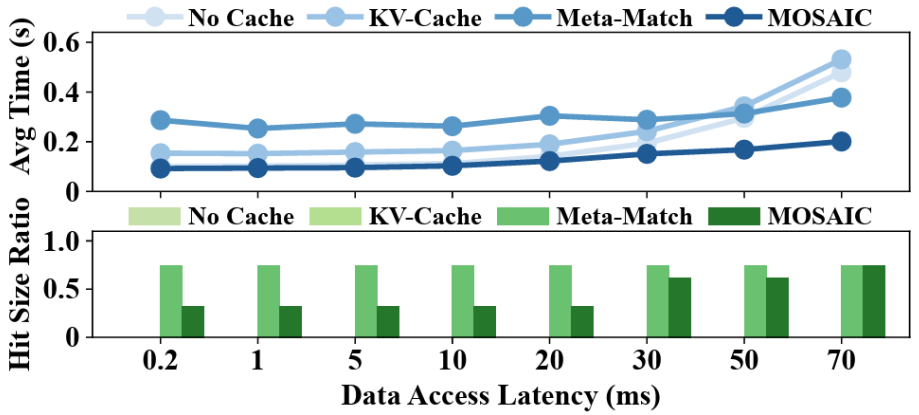


Figure 6: Impact of Data Access Latency.

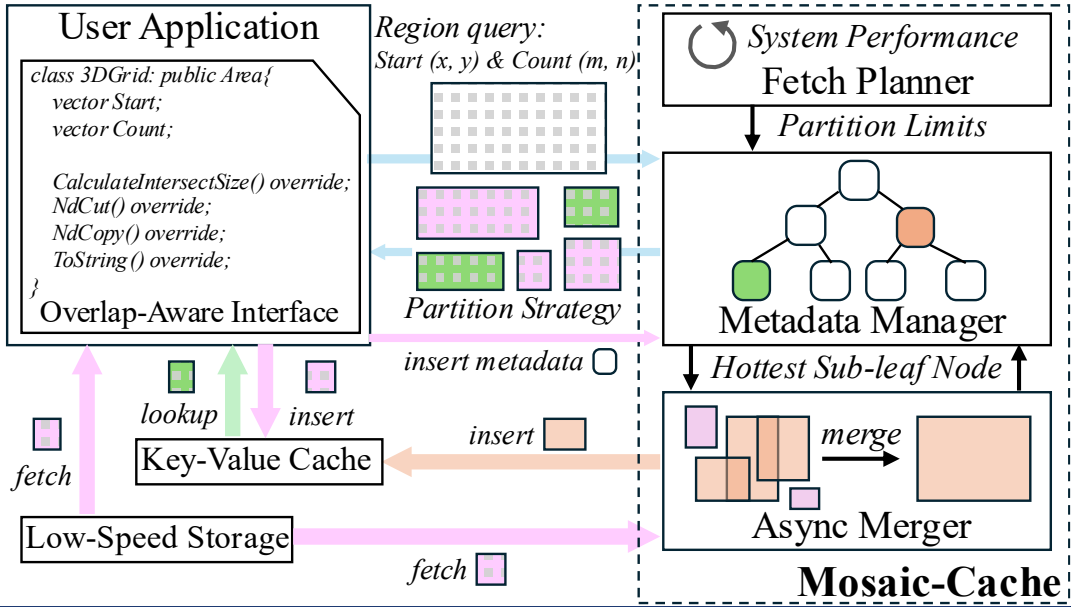
Conclusion and Future Work

Mosaic-Cache

- Enables content-level partial reuse
- Supports customizable caching components
- Achieves up to 4.1× speedup on real HPC datasets
- Minimal overhead in worst cases

Next Steps:

- Extend to popular application scenarios
- Evaluate with more real-world traces & datasets



Thank You!

Q & A

ASU-IDI

Contact

Chang Guo (cguo51@asu.edu)

Zhichao Cao (Zhichao.Cao@asu.edu)

https://asu-idi.github.io/contact/