Evaluating Performance Trade-offs of Caching Strategies for AI-Powered Querying Systems

Hyunju Oh[1], Wei Zhang[2], Christopher D. Rickett[3], Sreenivas R. Sukumar[3], Suren Byna[1]
[1]The Ohio State University, [2]Lawrence Berkeley National Laboratory, [3]Hewlett Packard Enterprise
oh.693@osu.edu, zhangwei217245@lbl.gov,
chris.rickett@hpe.com, sreenivas.sukumar@hpe.com, byna.1@osu.edu

2024 IEEE International Conference on Big Data (BigData) | 979-8-3503-6248-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BigData62323.2024.10825819

***Abstract*—With the rapid growth of accumulated data from various scientific domains, traditional data management systems face challenges in supporting complicated queries, such as pattern search, on massive amounts of data. To serve sophisticated queries through capturing precise features from data, recent data management systems seek to use artificial intelligence (AI) within the querying process. However, the characteristic of AI inference workflow within the querying process, such as intensive computation and expensive requirements for computing resources, becomes a bottleneck of the AI-powered query systems.**

In this paper, we provide a generalization of AI inference workflow in the context of AI-powered data discovery and we introduce three different caching strategies corresponding to each stage in the AI inference workflow. We provide in-depth performance evaluation on the impact of these caching strategies through a series of strong scaling experiments. Our experimental results show that the AI-powered data querying performance can be significantly improved by applying different caching strategies.

***Index Terms*—AI-powered Feature Querying, Caching**

I. Introduction

With the exponential growth of data in recent years, the development of efficient and effective data discovery methods has become necessary [1]–[5]. Traditional data discovery techniques, such as relational database management systems-based indexing and querying, struggle to keep pace with the increasing demands of modern massive volume datasets [6]. More importantly, the conventional querying methods fail to capture sophisticated patterns and features inherent in large datasets, limiting their ability to provide meaningful insights.

With the recent advancements in artificial intelligence (AI) and data management systems, some data discovery solutions with AI inference capability have emerged as powerful tools for extracting patterns and providing insights from massive datasets [7]–[10]. These solutions facilitate immediate data analysis and querying, thereby significantly enhancing the effectiveness of data discovery processes. In such a system [11], queries can contain user-defined filters utilizing AI inference to refine results based on specific ranges or patterns over the properties of objects in the graph.

However, with the AI inference process, these query systems face several challenges. First of all, the AI inference models usually incur intensive computations [12], [13], which can significantly slow down the query execution process. Secondly, many AI inference procedures may require expensive computing resources, including GPUs and memories, which may not always be affordable in various scenarios. In addition, due to the recursive characteristic of AI-powered query, the queries that are searching through multiple features can lead to repetitive raw data loading for the AI inference process, which can significantly strain the I/O bandwidth of the underlying storage system.

Facing these challenges in AI-powered query systems, it is essential to seek effective caching strategies that can help with boosting the performance of the AI-powered queries. Therefore, in this research, we propose different caching strategies based on the general steps involved in the AI inference process, and we aim to explore the impact of these caching strategies on the AI-powered query systems for performance improvement. Specifically, through this study, we seek to find answers to the following key questions:

- What data should be cached considering the AI inference workflow of the query systems?
- How does caching affect the overall performance of AI-powered queries?
- What are the merits and costs of different caching strategies for AI-powered queries?
- Which caching strategies are most effective for AI-powered queries?

Targeting these questions, our main contributions are:

Generalized View of AI Inference Workflow: we present a generalized view of AI inference workflow within the AI-powered query system considering the utilization of AI.

Caching Strategies for AI Inference Workflow: we introduce 3 different caching strategies that can be applied in an AI-powered query to balance performance improvement and dynamism of the querying system.

Thorough Evaluation of Caching Strategies: we evaluate the query performance in-depth providing thorough comparison between different caching strategies.

Insight from Observation: we present our observations on the impact of caching strategies for AI-powered queries and share insights regarding potential query performance improvements and associated trade-offs.

The rest of the paper is organized as follows. In Section II, we review related works of the AI-powered query. Then, in Section III, we identify the AI inference workflow and propose different cache levels for mitigating the AI inference overhead in the query processing stage. We show our experimental results in Section IV. Finally, we conclude in Section V.

II. Background

In this section, we will review the AI-powered query for scientific data discovery and related works.

A. The Need of AI-powered Data Discovery

The application of artificial intelligence is nowadays revolutionizing various domain sciences such as biometric authentication, biodiversity, medical, and astronomy research, etc [14]–[20]. With the widespread application of AI in domain sciences, there is a growing need for data discovery with a focus on particular features of interest, especially when the features are dynamically captured by AI inference process [19], [21]–[23]. While many traditional database management systems [24]–[29] do not directly provide data query capability that captures the underlying features or patterns in the data, in the past few years, several solutions, including MADlib [30], Spark SQL [10], and Cray Graph Engine (CGE) [31], are emerging with the support of AI-powered data querying. In these solutions, the query execution process will include a dynamic data filtering step implemented as a user-defined function (UDF) where relevant data samples are selected based on the features of interest that are automatically captured by ML/DL models in real-time. This practice highlights the innovative use of databases with UDF features, indicating great potential for AI-powered data discovery.

However, there are two major challenges in these systems to provide AI-powered query capabilities. The AI-inference process is known to require significant computing power, and generally it is a time-consuming process iterating over multiple phases [10], [32]. These drawbacks consequently result in higher resource requirements for the machines that can host such systems, which hinders the prevalence and maintainability of these systems. Furthermore, the repetitive characteristic of AI inference leads to multiple rounds of data loading, and this can cause excessive data loading within the real-time AI-powered query processing which results in increased query duration. Therefore, there is a need to accelerate the AI-powered query processing, minimizing the resource requirement of such system, and to avoid excessive data loading that can overwhelm the underlying storage system.

B. Data Pipelines of AI-powered Data Discovery

To understand how AI-powered data discovery works, we study the data pipeline of AI-powered data discovery in Cray Graph Engine (CGE) [11]. Since CGE combines high-performance graph processing with AI inference, we consider it as a well-suited example for demonstrating the scalability and efficiency of AI-powered data discovery systems.

1. Data Ingestion: The data ingestion process begins with loading input data, typically large and unstructured, into the system. During ingestion, the user-defined load operations are executed by the ranks in the CGE instance. These operations are custom user programs designed to extract relevant features or relationships from the input data. Once the user program completes, it returns edges representing these features, which are then inserted into the graph structure for future queries.

2. Query Execution: This stage consists of accepting an input query from the user, which in the case of CGE is written in SPARQL, translating the query to a set of operations (e.g., scan, join and filter) and coordinating amongst the parallel processes to execute the operations. In query execution of CGE, the UDFs enable users to apply domain specific AI models to capture the features of the data owned by each rank, and compare it with the query condition in the specified filter of the given query.

III. Caching Strategies in the AI Inference Workflow of AI-powered Data Discovery

In this section, we will review the AI inference workflow in AI-powered data discovery and elaborate on the caching strategies corresponding to different steps in the AI inference workflow.

A. AI Inference Workflow

In real-world applications, to achieve optimal result, the AI inference process usually utilizes a pipeline of structured, focused, and refined AI inference steps. Typically, these steps can be generalized as 3 major stages as illustrated in Figure 1, including 1) **Object Extraction (i.e., OE)**, 2) **Target Identification (i.e., TI)**, and 3) **Feature Extraction (i.e., FE)**. While the last two can sometimes be optional, in AI-powered data discovery where query conditions are specified to hit the features of the objects, we consider that these 3 steps are generally applicable in the AI-powered data discovery process. Such practice also ensures the modularity of AI inference process, making it possible to reuse the AI inference result at certain step in the process. Here, we present each stage :

*1) **Object Extraction (OE)**:* The first stage in the AI inference workflow is object extraction, where the system identifies and extracts relevant objects from the raw data. This approach significantly improves efficiency by reducing unnecessary computation and ensuring that subsequent stages work only on relevant data. The benefits of having an object extraction stage are manifold. First, it significantly reduces the complexity of the data processing pipeline by allowing subsequent stages to work on recognizable objects, rather than being overwhelmed by noises in the background. As illustrated in the crime analysis application in Figure 1, in the entire AI inference workflow that aims to extract the age and gender feature of potential criminals in the video frames of surveillance footage, the object extraction stage is design to first extract the recognizable objects such as people, sofa, and bookshelf. Hence, for generally collected datasets, retrieving the data containing the desired objects is necessary. Additionally, object extraction enables modularity and reuse, as once the bounding boxes and object information are stored, they can be reused for different queries or analyses without having to recompute or reprocess the raw data. This not only increases the efficiency of AI-powered queries but also supports flexibility in multi-query environments, enabling faster response times and more dynamic data discovery.

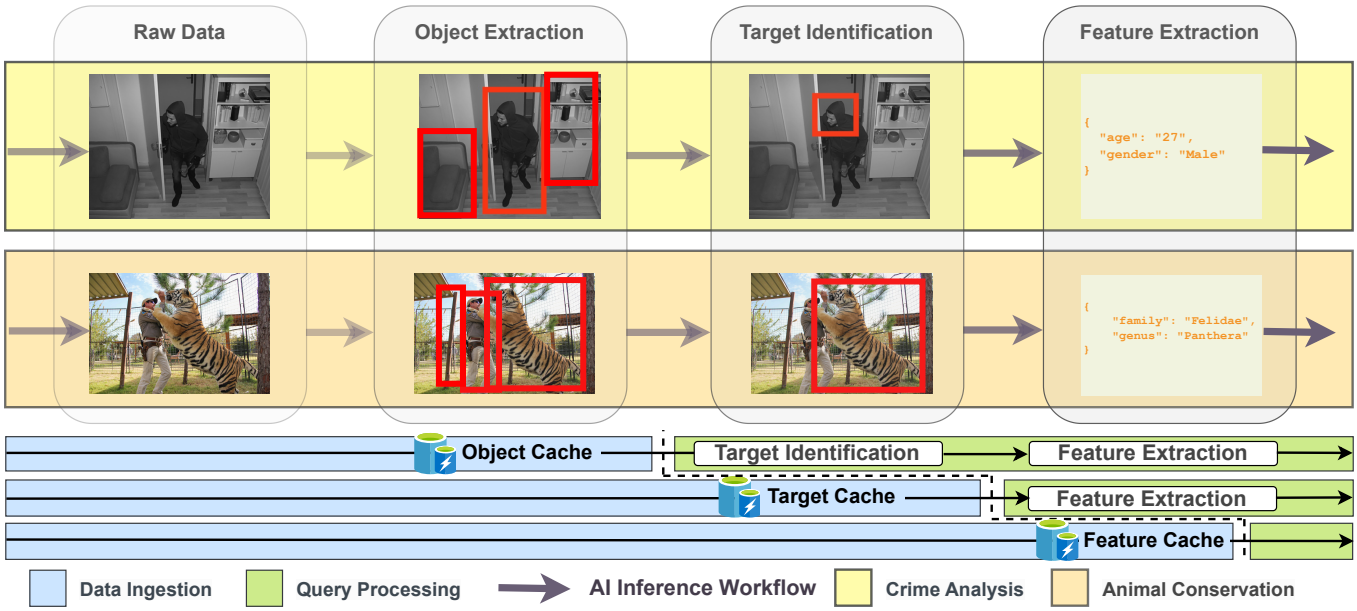

Figure 1. An overview of AI inference workflow and caching strategies

*2) **Target Identification (TI):*** With the partial image of the objects, the second stage of AI inference workflow is to identify targets of interest among these objects to further narrow down the system's attention to the specific objects that are most likely to meet the requirements of the query. As depicted in Figure 1, in crime analysis, particularly in facial recognition systems, this stage helps the system focus on human faces extracted from video frames. It eliminates non-human objects and ensures that the subsequent analysis is carried out only on faces, which is essential for tasks like identifying potential suspects. Similarly, in animal conservation, when searching for a specific species, this step enables the system to isolate the relevant animals from the Felidae family, ignoring other elements like people or plants in the environment. Target identification ensures that only relevant data passes to the next stage, making the entire process more efficient and focused.

This step is crucial since it significantly refines the intermediate results, ensuring that only relevant objects proceed to the next step for further analysis. By filtering out unrelated objects, such as furniture when searching for people or identifying animal instead of humans when searching for species, the system can perform AI inference with better efficiency and accuracy. While this step is not always mandatory, in AI-powered data discovery workflows, target identification greatly enhances the relevance of the results. However, since this stage still involves AI inference that is computationally expensive, when the system responds to huge amount of queries, it will also play a crucial role in straining the resources and slowing down the queries.

*3) **Feature Extraction (FE):*** The necessity of the feature extraction stage comes down to the requirement that the users should be able to find data objects that have certain features. Since the query condition against the feature is usually in the form of text, this step typically involves the procedure that performs AI inference on top of the relevant objects identified by the previous stage and generates descriptive feature labels. For example, in the crime analysis scenario, after human faces are identified, the feature extraction stage will run specific AI models to identify properties and generate descriptive labels such as "gender: Male" and "age: (25-32)". Similarly, in the animal conservation example, based on the identified animals, the feature extraction stage will perform animal taxonomy inference to generate species taxonomy labels, such as "family: Felidae" and "genus: Panthera".

This stage transforms identified targets of interest into searchable insights that can be queried. Without feature extraction, user will only be able to locate objects of broad categories, such as "people" and "animal", but won't be able to refine their search based on the attribute labels such as age and gender, or family and genus. However, depending on the AI models in this stage, the computation can still be intensive and hence strains computing powers and degrades the query performance.

B. Caching Strategies

Given the three stages of the AI inference workflow, we consider caching objects, identified targets, and extracted features strategically, so that we can minimize the overhead associated with repetitive AI inference steps, thereby accelerating the overall query execution while maintaining precision. Specifically, these caching strategies are:

*1) **Object Caching:*** Object caching refers to the process of storing the results of the object extraction stage in the AI inference workflow. With this, the extracted objects can be reused for future queries without repetitively running the object extraction procedure. In CGE, although the bounding

boxes of the identified objects are already stored as metadata of the objects, the object extraction step in the AI-powered query execution still need to perform raw image file loading, image cropping, and other necessary preprocessing operations, such as RGB channel transformations, necessary resolution adjustment, etc. While these operations are less expensive than re-running the object identification model, they can still be resource-intensive, particularly when the raw image files are large. By storing the preprocessed objects, future queries can bypass the object extraction stage by directly loading the cached objects.

*2) **Target Caching**:* Target caching is a strategy that stores the results of the target identification process, where the system determines whether the object data contains the specific targets required by the query. In AI-powered data discovery, target identification is crucial because it filters relevant objects based on the query's focus, such as faces in crime analysis or specific animals in animal conservation scenario. By caching this target data, applications can bypass the repetitive and often resource-intensive process of target identification, directly retrieving objects that match the specified targets. This significantly reduces the computational load during subsequent queries, allowing the system to efficiently move to the feature extraction stage without unnecessary delays.

*3) **Feature Caching**:* Feature caching focuses on storing the results obtained from the feature extraction process in AI-powered queries. The ultimate goal for the AI-powered query is to retrieve data containing specific features, such as age, gender, or species classification, which are extracted through the AI inference models. By caching the extracted features, the system can skip the entire pipeline - object extraction, target identification and feature extraction - by directly loading the cached feature matching with the query condition. This can lead to significant performance gains.

*4) **Necessity of Each Caching Strategy**:* While feature caching seems to provide the maximum benefit by allowing the query execution bypassing the entire AI inference pipeline, it is not always sufficient on its own. Object caching and target caching remains valuable when queries are not only asking about the fully processed feature, but sometimes even the information in the intermediate results. For instance, when new data is constantly ingested, features may not be immediately available in the cache. In such case, object caching and target caching still can help with accelerating the AI inference workflow and hence the entire query execution pipeline. Moreover, new attributes or features can always be introduced into the system that were not part of the original feature extraction process. For example, if original cached facial features only include age and gender, and the new query requires extracting a different feature like facial expression, the system would need to go back to the object and target data to perform new feature extraction tasks. Therefore, all the caching strategies we have discussed here can be useful depending on the performance, flexibility and scalability requirements of the querying system.

IV. Evaluation

To evaluate the impact of different caching strategies on AI inference and AI-powered data queries, in this section, we show our evaluation result and provide analytical discussions.

A. Experimental Setup

1) Platform and Dataset: The experiments are conducted to show the impact of three different caching strategies on the AI-powered query performance. By comparing the performance with the default query system where no caching stategy is applied, we also evaluate the impact of different caching strategies on the corresponding stages in the AI inference workflow. Particularly, we run the experiments on two different environments to present the effect of caching strategies and performance enhancement when combined with scaling in parallel environments: 1) multi-process scaling experiment and 2) multi-node scaling experiment. Each of the experiments are executed on two different machines of HPE Cray EX supercomputer using CPU nodes with specification of dual-socket AMD EPYC 64-core processors. In the multi-process scaling experiment, we fix the number of nodes to 4 and increase the number of processes per node from 8 to 32. In the multi-node scaling experiment, we scale the number of nodes from 4 to 128 and fix the number of processes per node to 16. The dataset we use in this experiment contains $\sim$ 90,000 images extracted from Google OpenImages datasets [33] using openimages python package. Software-wise, we use the Cray Graph Engine [11] as the data platform to test both data ingestion and query execution performance.

To demonstrate the general applicability of the caching strategies, we applied two use cases which are the facial recognition use case previously mentioned in the crime analysis scenario and the animal taxonomy use case from the aforementioned animal conservation application. For the object extraction inference process, both use cases go through the same inference model using torchvision models [34]. For target identification inference process and feature extraction inference process, the two use cases used different inference models to extract domain-specific features, specifically, for facial recognition, we use the face detection model for target identification and the age/gender detection models [35], [36] for feature extraction; for animal taxonomy, we use Mobilenet_V2 model in the torchvision package [34] to perform target identification process for identifying animals, and we use BioCLIP model [37], [38] to perform feature extraction, capturing the animal taxonomy features.

2) Caching Strategy Implementation: To apply the caching strategies within the AI inference workflow of CGE and to show the impact on the AI-powered query performance, we have implemented caching within the data ingestion workflow which is executed prior to query execution. For each caching levels, the data ingestion process will additionally go through necessary AI inference steps, and cache the data by generating an edge within the in-memory database. Each of the caching strategies are implemented as following:

Object Caching: In the object extraction stage of data ingestion, after the objects are detected in the data ingestion process, the objects cropped from the image will be processed to be ready for target identification stage. When we choose object caching strategy, we store the result of this stage into CGE graph as a property edge of the corresponding object vertex in the knowledge graph.

Target Caching: Subsequent to the object extraction process, the target caching strategy will additionally go through the target identification step within the data ingestion process. Using the object data, it will perform inference for targets and the detected targets will be stored as a property edge of the corresponding object vertex in the knowledge graph.

Feature Caching: For the feature caching strategy, the data ingestion workflow will go through both object extraction and target identification step, and proceed onto feature extraction to acquire feature labels. The list of feature labels will be stored as a property edge of the corresponding object vertex in the knowledge graph.

3) Experiment Procedure: For both multi-process scaling and multi-node scaling experiment, the experiment procedure follows the same steps. First, before executing the AI-powered query, we perform the data ingestion process to scan the image data and to generate the knowledge graph managed by CGE. Depending on the caching strategy we choose in our experiment, we extend data ingestion process to the corresponding stage of the AI inference workflow and cache the result of that stage. Upon completion of the data ingestion process, the knowledge graph in CGE is readily built for AI-powered query.

```
select distinct ?src ?obj1 ?obj2
where {
    ?img a <urn://image>;
        <urn://source> ?src ;
        <urn://contains> ?obj1 ;
        <urn://contains> ?obj2 .
    ?obj1 <urn://bounding-box> ?box1 .
    ?obj2 a "dog" ;
        <urn://bounding-box> ?box2 .
    filter(arq:py_user_func('feature_query', 'feature_query_func',
                            ?src, 'age', ?box1) = '(25-32)')
}
```

(a) Facial Recognition

```
select distinct ?img ?src ?obj1
where {
    ?img a <urn://image>;
        <urn://source> ?src ;
        <urn://contains> ?obj1 ;
        <urn://contains> ?obj2 .
    ?obj1 <urn://bounding-box> ?box1 .
    ?obj2 a "zebra";
        <urn://bounding-box> ?box2 .
    filter(arq:py_user_func('feature_query', 'feature_query_func',
                            ?src, 'family', ?box1) = 'Felidae')
}
```

(b) Animal Taxonomy

Figure 2. Sample AI-powered SPARQL Query Statement.

With the knowledge graph constructed during the data ingestion process, we execute the AI-powered query experiment. To avoid being affected by the potentially cached query result,

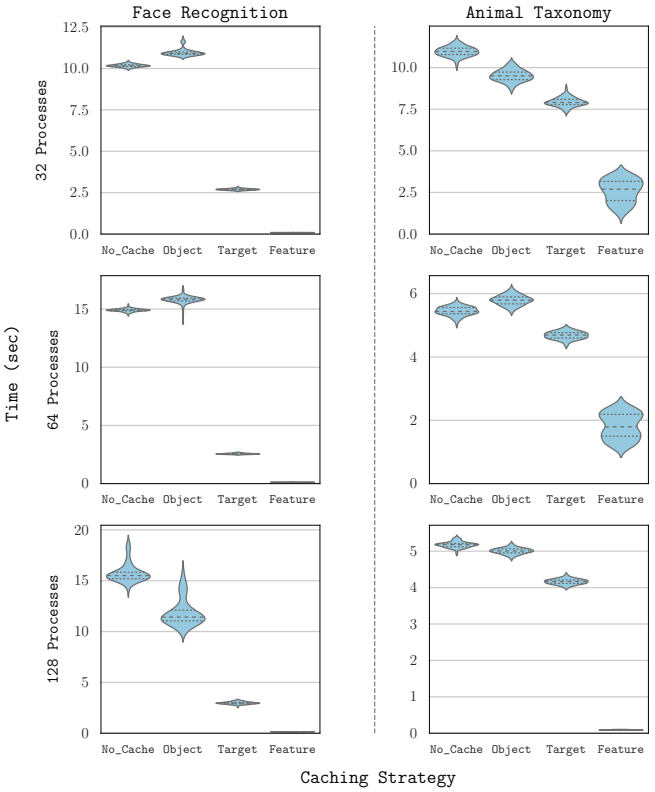

Figure 3. The query latency for multi-process scaling experiment. Each row represents the number of processes used for the experiment, and each column represents the use case that was executed. For each plots, it presents the distribution of execution time when using different caching strategy.

we use a different query condition for each query we issue to CGE. This will allow us to accurately evaluate the actual query time performance. As shown in Figure 2, for the facial recognition use case, we will randomly use either age or gender features for `?obj1` in our query. For animal taxonomy, we will randomly use either family or genus features for `?obj1` in each query.

B. Multi-process Scaling Experiment

We conduct our multi-process scaling experiment on 4 compute nodes on Perlmutter supercomputer at NERSC. We would like to observe how different caching strategy affect the overall query performance and also what happens to each stage in the AI inference workflow. Also, we would like to see with a fixed number of nodes, what happens when we increase the total number of processes.

1) Query Latency: The overall performance where we measured the elapsed time for each execution of query is shown in Figure 3. For all 6 settings (2 use cases and 3 process count settings), we can see the performance improvement when target caching and feature caching are applied, as compared to the default no cache setting in the system. When scaling the number of processes, each use case showed a different trend in the query performance. For the facial recognition use case, using 64 processes and 128 processes showed decreased query

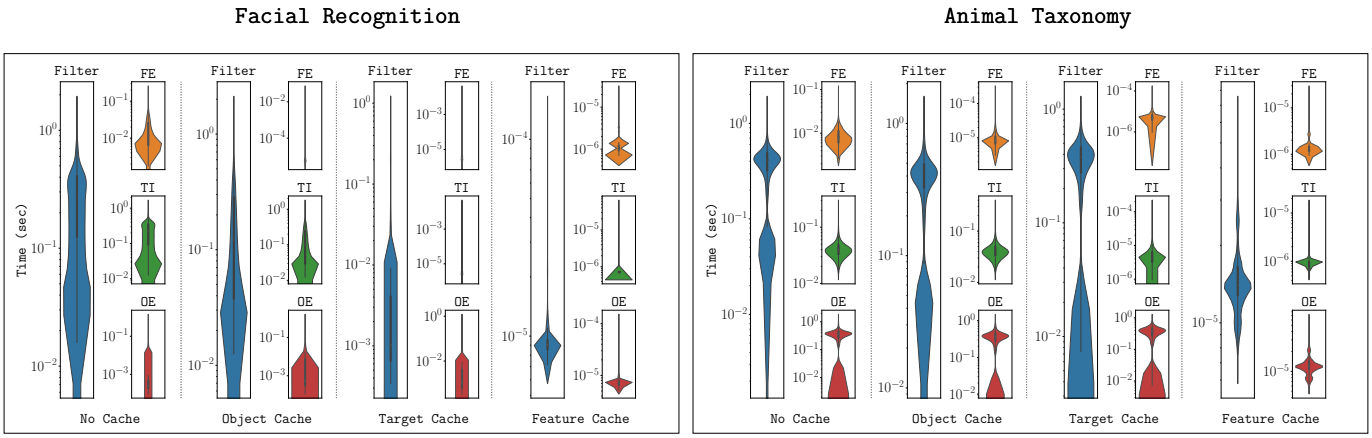

Figure 4. Breakdown of AI Inference Execution Time

performance than using 32 processes. On the other hand, with the animal detection use case, as we increased the number of processes, it showed a performance improvement. In general, for smaller number of processes, the performance of object caching strategy becomes unstable, while for large number of processes, the object caching strategy remains effective for reducing the query latency. As previously mentioned, when applying the object caching strategy, it will only replace a series of operations where no AI inference is performed, such as image cropping, RGB channel tranformation, resizing, etc. While these operations can be easily handled by modern CPU without much inter-process communication and I/O overhead, the cost of reading cached content from device may sometimes outweigh the overhead of operations that were replaced.

2) Breakdown of AI-Inference Filter Latency: To analyze the impact of caching strategies on the various steps within the AI inference workflow, we present the execution time of object extraction, target identification, and feature extraction across all queries in our 128-process test in 2 use cases on 4 nodes, as shown in Figure 4.

In Figure 4, for each use case, there are four blue violin plots which present the distribution of total AI inference time per image for the different caching strategies, as shown in the "Filter" columns in each use case. Each of them presents the execution time distribution of the UDF filter used in the query as shown in Figure 2. On the right side of each blue violin plot, we provide detailed elapsed time distribution for the Object Extraction (OE), Target Identification (TI), and Feature Extraction (FE) steps in the AI inference UDF, shown in red, green, and orange violin plots.

Overall, we can see that, in both use cases, the total AI inference time ("Filter") is decreasing each time when we move on to the next caching level, which clearly demonstrate how different caching strategies help with reducing the execution time of AI Inference UDF in the query. Specifically, we can clearly observe that the execution time of the TI and FE steps significantly decreased at the correspoding cache level since the cached data can be directly used without performing the actual AI inference for those steps. This indicates two key findings related to the AI-powered query performance:

- The performance of AI-powered query is significantly affected by the AI inference workflow.
- The adoption of caching strategies within the AI-powered query can improve the performance AI inference workflow.

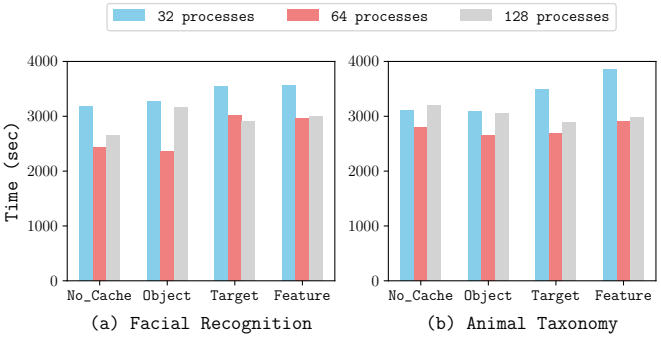

Figure 5. Data Ingestion Time in Multi-process Scaling Experiment

3) Evaluation of Data Ingestion Performance: Figure 5 reports the data ingestion time. For both facial recognition and animal taxonomy use cases applying four different caching levels, there was an improvement in performance by scaling the number of processes from 32 processes to 64 processes. However, from 128 processes, it shown reduction of performance compared to 64 processes configuration. This shows that for the fixed number of node configuration with fixed number of cores, the scaling of processes would have a limitation in improving the data ingestion workflow. In this case, for the 4 node configuration, scaling the number of processes up to 64 would enhance the data ingestion with four caching levels, but from 128 processes, it would degrade the performance, most likely due to contention on resources, such as memory and cache.

In Figure 6, we present a more detailed timing performance regarding where the overhead is occurring within the data ingestion workflow. For the use cases we selected, we can see

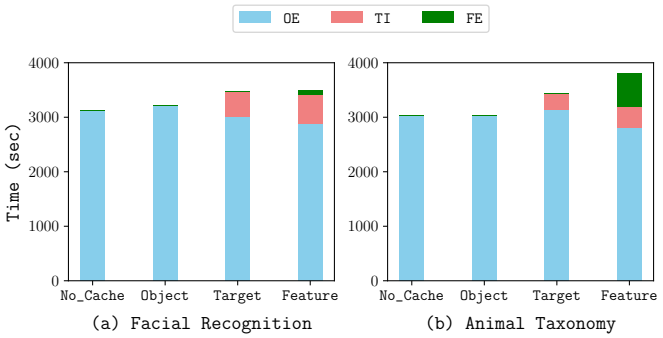

Figure 6. Breakdown of Data Ingestion Time

a slight growth in the data ingestion time for target caching and feature caching strategies, as these would require additional AI inference steps to generate the target identification and feature extraction results to be cached. However, we can see that, as compared to the execution time of object extraction time, the execution time for target identification and feature extraction is negligible, especially considering the fact that data ingestion for every piece of raw data is a one-time operation, while the cache stored in this process can help with millions or even billions of queries, applying these caching strategies can be very rewarding.

C. Multi-node Scaling Experiment

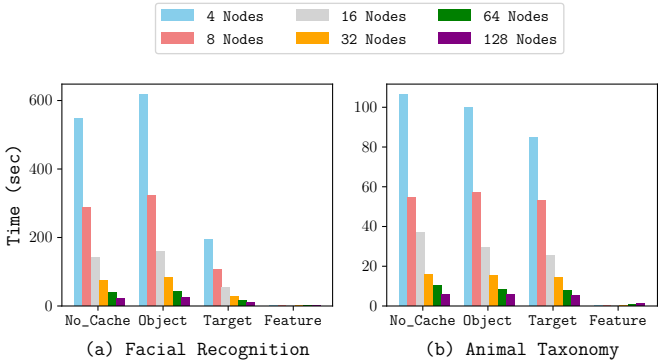

Figure 7. Query Latency of Multi-node Scaling Experiment

In the multi-node scaling experiment, we present the performance of the AI-powered query and data ingestion by scaling the number of nodes. Through this experiment, we will focus on the impact on the AI-powered query when different caching strategies are combined with node-level scaling.

1) Query Performance: In Figure 7, we present the elapsed time per query as we increase the number of nodes from 4 to 128 and use 16 ranks per node (i.e., 64 to 2048 total ranks across nodes). As the number of nodes increase, we were able to see a linear performance improvement for each caching strategies. In addition, for both use cases, the application of target caching and feature caching showed consistent performance improvement similar to result of multi-process scaling experiment. The feature caching performance

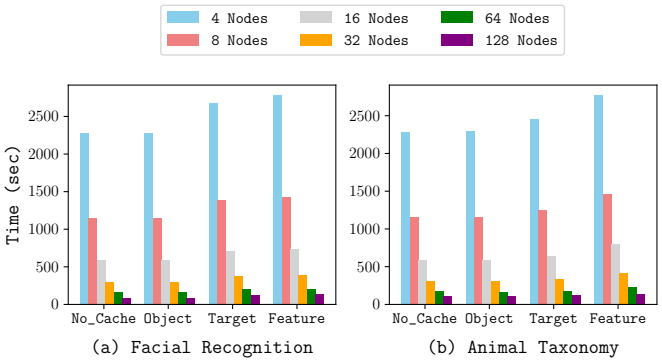

Figure 8. Data Ingestion Time of Multi-node Scaling Experiment

for all node counts was orders of magnitudes faster than all other caching strategies. For example, on 32 nodes, one set of the facial recognition query times for no cache, object cache, target cache and feature cache were 74.1, 82.4, 28.9 and 0.42 seconds, respectively. These results clearly demonstrate how the combination of appropriate selection of caching strategy and number of nodes can significantly accelerate the speed of AI-powered queries. This shows that the caching strategies, especially target caching and feature caching, can generally help with improving the query latency of AI-powered data discovery across various scales.

2) Data Ingestion Performance: The data ingestion performance is shown as Figure 8. We present the overall time it took for generating the required data for each caching strategy as the number of node scales. For each caching strategy, as similar to the query performance, we observed a linear time performance enhancement as we increase the number of nodes. Hence, we were able to observe that the overhead within the data ingestion process can be significantly reduced if unified with scaling of nodes. This shows that the additional cost of applying caching strategies can be further reduced when the number of nodes increases.

V. Conclusion

In this study, we present three caching strategies to help with reducing repetitive yet time-consuming AI inference operations in the context of AI-powered data querying. Our experiment result offer in-depth comparisons of the effect of these caching strategies. Overall, our study shows that, at different stages of AI inference workflow, these caching strategies are generally effective for reducing the time-consuming AI inference operations, and can be adopted opportunistically in different scenarios depending on the reusability of the cached content. While the trade-offs of utilizing these caching strategies is the additional AI inference time spent in the data ingestion process which only occur once, countless queries can benefit from the acceleration these caching strategies offer, which demonstrates the value of caching strategies in the AI-powered data discovery.

REFERENCES

[1] D. R.-J. G.-J. Rydning, J. Reinsel, and J. Gantz, "The digitization of the world from edge to core," *Framingham: International Data Corporation*, vol. 16, pp. 1–28, 2018.

[2] P. K. Sadineni, "Comparative study on query processing and indexing techniques in big data," in *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*. IEEE, 2020, pp. 933–939.

[3] W. Zhang, S. Byna, H. Tang, B. Williams, and Y. Chen, "MIQS: metadata indexing and querying service for self-describing file formats," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019*, M. Taufer, P. Balaji, and A. J. Peña, Eds. ACM, 2019, pp. 5:1–5:24. [Online]. Available: https://doi.org/10.1145/3295500.3356146

[4] W. Zhang, H. Tang, S. Byna, and Y. Chen, "Dart: distributed adaptive radix tree for efficient affix-based keyword search on hpc systems," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018, pp. 1–12.

[5] W. Zhang, H. Tang, and S. Byna, "Idioms: Index-powered distributed object-centric metadata search for scientific data management," in *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2024, pp. 598–608.

[6] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, "Comparison between relational and nosql databases," in *2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO)*. IEEE, 2018, pp. 0216–0221.

[7] R. Kashyap, "Machine learning in google cloud big query using sql," *SSRG International Journal of Computer Science and Engineering*, vol. 10, no. 5, pp. 17–25, 2023.

[8] A. Kulkarni, "Amazon athena: Serverless architecture and troubleshooting," *International Journal of Computer Trends and Technology*, vol. 71, no. 5, pp. 57–61, 2023.

[9] M. Kachuee, A. Hosseini, B. Moatamed, S. Darabi, and M. Sarrafzadeh, "Context-aware feature query to improve the prediction performance," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2017, pp. 838–842.

[10] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1383–1394.

[11] C. D. Rickett, K. J. Maschhoff, and S. R. Sukumar, "Massively parallel processing database for sequence and graph data structures applied to rapid-response drug repurposing," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 2967–2976.

[12] A. N. Mazumder, J. Meng, H.-A. Rashid, U. Kallakuri, X. Zhang, J.-S. Seo, and T. Mohsenin, "A survey on the optimization of neural network accelerators for micro-ai on-device inference," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 532–547, 2021.

[13] S. Nakandala, K. Nagrecha, A. Kumar, and Y. Papakonstantinou, "Incremental and approximate computations for accelerating deep cnn inference," *ACM Transactions on Database Systems (TODS)*, vol. 45, no. 4, pp. 1–42, 2020.

[14] P. Kaur, K. Krishan, S. K. Sharma, and T. Kanchan, "Facial-recognition algorithms: A literature review," *Medicine, Science and the Law*, vol. 60, no. 2, pp. 131–139, 2020.

[15] M. Tan, W. Chao, J.-K. Cheng, M. Zhou, Y. Ma, X. Jiang, J. Ge, L. Yu, and L. Feng, "Animal detection and classification from camera trap images using different mainstream object detection architectures," *Animals*, vol. 12, no. 15, p. 1976, 2022.

[16] A. Sharma and P. K. Mishra, "Performance analysis of machine learning based optimized feature selection approaches for breast cancer diagnosis," *International Journal of Information Technology*, vol. 14, no. 4, pp. 1949–1960, 2022.

[17] P. Danaee, R. Ghaeini, and D. A. Hendrix, "A deep learning approach for cancer detection and relevant gene identification," in *Pacific symposium on biocomputing 2017*. World Scientific, 2017, pp. 219–229.

[18] Z. Hu, J. Tang, Z. Wang, K. Zhang, L. Zhang, and Q. Sun, "Deep learning for image-based cancer detection and diagnosis- a survey," *Pattern Recognition*, vol. 83, pp. 134–149, 2018.

[19] K. Choudhary, B. DeCost, C. Chen, A. Jain, F. Tavazza, R. Cohn, C. W. Park, A. Choudhary, A. Agrawal, S. J. Billinge *et al.*, "Recent advances and applications of deep learning methods in materials science," *npj Computational Materials*, vol. 8, no. 1, p. 59, 2022.

[20] R. E. González, R. P. Munoz, and C. A. Hernández, "Galaxy detection and identification using deep learning and data augmentation," *Astronomy and computing*, vol. 25, pp. 103–109, 2018.

[21] A. Nadeem, M. Ashraf, K. Rizwan, N. Qadeer, A. AlZahrani, A. Mehmood, and Q. H. Abbasi, "A novel integration of face-recognition algorithms with a soft voting scheme for efficiently tracking missing person in challenging large-gathering scenarios," *Sensors*, vol. 22, no. 3, p. 1153, 2022.

[22] K. Park, K. Saur, D. Banda, R. Sen, M. Interlandi, and K. Karanasos, "End-to-end optimization of machine learning prediction queries," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 587–601.

[23] J. Ye, J. Hai, Z. Wang, C. Wei, and J. Song, "Leveraging natural language processing and geospatial time series model to analyze covid-19 vaccination sentiment dynamics on tweets," *JAMIA open*, vol. 6, no. 2, p. ooad023, 2023.

[24] S. Sanfilippo, "Redis," https://redis.io, 2009, version 6.0, Last accessed: 2024-09-06.

[25] M. D. Team, "Mysql," https://www.mysql.com, 1995, version 8.0, Last accessed: 2024-09-06.

[26] I. Facebook, "Rocksdb," https://rocksdb.org, 2013, version 6.0, Last accessed: 2024-09-06.

[27] I. MongoDB, "Mongodb," https://www.mongodb.com, 2009, version 6.0, Last accessed: 2024-09-06.

[28] D. R. Hipp, "Sqlite," https://www.sqlite.org, 2000, version 3.42, Last accessed: 2024-09-06.

[29] P. G. D. Group, "Postgresql," https://www.postgresql.org, 1996, version 14.0, Last accessed: 2024-09-06.

[30] J. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li *et al.*, "The madlib analytics library or mad skills, the sql," *arXiv preprint arXiv:1208.4165*, 2012.

[31] C. D. Rickett, U.-U. Haus, J. Maltby, and K. J. Maschhoff, "Loading and querying a trillion rdf triples with cray graph engine on the cray xc," *Cray User Group*, 2018.

[32] N. N. Alajlan and D. M. Ibrahim, "Tinyml: Enabling of inference deep learning models on ultra-low-power iot edge devices for ai applications," *Micromachines*, vol. 13, no. 6, p. 851, 2022.

[33] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Malloci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy, "Openimages: A public dataset for large-scale multi-label and multi-class image classification." *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[34] T. maintainers and contributors, "Torchvision: Pytorch's computer vision library," https://github.com/pytorch/vision, 2016.

[35] G. Levi and T. Hassner, "Emotion recognition in the wild via convolutional neural networks and mapped binary patterns," in *Proceedings of the 2015 ACM on international conference on multimodal interaction*, 2015, pp. 503–510.

[36] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[37] J. Bradley, H. Lapp, and E. G. Campolongo, "pybioclip," Jul. 2024.

[38] S. Stevens, J. Wu, M. J. Thompson, E. G. Campolongo, C. H. Song, D. E. Carlyn, L. Dong, W. M. Dahdul, C. Stewart, T. Berger-Wolf, W.-L. Chao, and Y. Su, "BioCLIP: A vision foundation model for the tree of life," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 19 412–19 424.