FULL REPLY TO REVIEWS

1 RESPONSE TO REVIEWER COMMENTS

Thanks for your careful reviewing. The comments help us to further improve the quality of this paper, we are very grateful to have the chance to reply these helpful reviews and clarify some misconceptions. Hopefully, the detailed responses can answer your questions and improve the reading experience.

TO REVIEWER #1

O1: The presentation is hard to follow and understand.

Reply: We are sorry for poor writing and sorry for confusing you. We are deeply grateful for your patience. We plan to rewritten or remove all of the bad examples, notations and definitions in next version of the paper to improve the presentation. The important questions are replied in the following.

(1)-(2) The presentation on the counter index and the skip tree is unclear. Please use a single example with concrete data values. Please show the data values in all the structures, *e.g.*, stored columns and the skip trees. The description mentions CORES[49] many times. It may be better to describe CORES as a baseline first ?

Reply: Thanks for your helpful advice. We are sorry for poor presentation that confusing you. We plan to adjust the structure of the paper in next version, add system overview to show the overall designs and data values in all structures and their relations as shown in Figure 1 . We may also add background chapter to briefly summarize some basic designs and baseline models such as CORES[49] as you mentioned. However, CORES is designed toward nested records only and additional data modeling and coding are needed to handle the cross-model analysis. We have made lots of efforts to extend the embedding scheme and bitset-based filtering pushdown to cross-multi-model analysis scenario, develop the unified columnar storage for multi-model data and a novel skipping scheme for cross-model queries evaluation.

(3) Tree seems to mean several things in the paper. For example, it can be the schema tree that contains only the attribute names and types. It may contain the data instances. The skip-tree index is also a tree. The presentation is confusing sometimes because of this. In Algorithm 1, T is tree schema, which seems to suggest that T contains only attribute names and types. However, the algorithm visits the counter array from nodes in T, which is associated with data. The terms level and layer are confusing. They seem to mean either the level in the schema tree or the different component in the skip tree. Please clarify the description.

Reply: Thanks for your careful reviewing. First, we would like to apologize for confusing you due to the careless definition. As the title of section 2.2 "Unified Tree Metadata Management", we view the Counter and Indicator arrays that maintain the mapping information as metadata and organize as a metadata tree structure. The metadata tree contains the structural information in the form of Counter and Indicator arrays, which is parallel to the data instance columns. And the precomputation is performed on the metadata tree, which constructs the skip-tree index denoted as ST in algorithm 2. We neglected the difference between schema tree and metadata tree in algorithm 1. As you mentioned, the schema

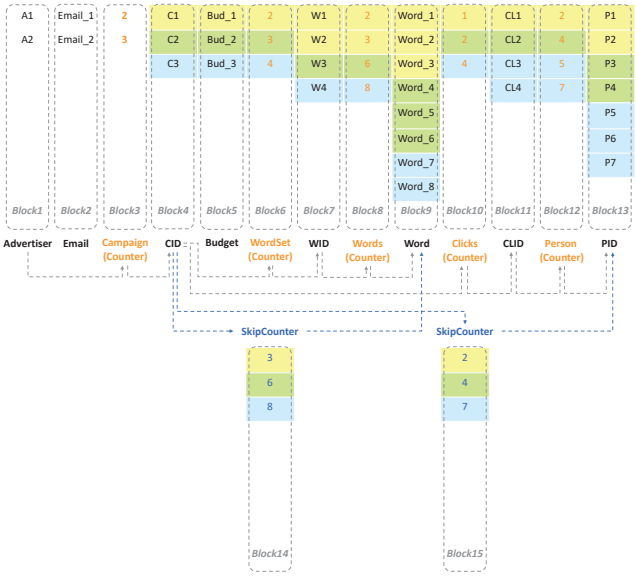

Figure 1: Illustration for pure columnar data layout

tree should contains only attribute names and types, the metadata tree is the correct input of algorithm1 and we should assign a new symbol for it e,g, *MT*, we are sorry again for confusing you. As for the input tree of algorithm 3, it could be the basic schema tree T since this algorithm doesn't need to invoke the mapping information maintained by metadata. Besides the statement of algorithm 1, we will take a close examination to eliminate the ambiguity and misconceptions caused by our poor writing and presentation.

(4) In Sec 3.1, what query types are supported? Is there a volcano-style query operator tree? Do you assume conjunctive predicates (because only the and operation is performed)? The description uses operators and nodes. Are these the same? It seems to me that the query evaluation does not use any relational algorithms, but rather performs tree traversal. Please give an example query and describe how the query is evaluated in the design using a figure.

Reply: Thanks for your careful reviewing again. Since QUEST is mainly developed towards scan-intensive cross-model analysis, its optimized query evaluation components such as bitset-based query pushdown and cross-model payload delivery mainly focus on the conditional queries and cross-model join queries. However, these design principles do not conflict with other analytical queries like aggregation, sorting and grouping, etc. As mentioned in CORES[49], these operator processing cost on top of highly selective filters is relatively much smaller than the I/O cost caused by data scans in disk-based storage systems, thus we omitted the specific analysis in paper. However, the proposed query evaluation scheme can be seamlessly applied to other column-oriented systems to optimize the processing of scan and join operators so as to better support other types of analytical queries on top of highly selective filters.

And there is no a volcano-style query operator tree, QUEST is proposed as a novel query evaluation scheme towards scan-intensive cross-model analysis but not a mature storage engine, we are still integrating other query execution plan compilation and optimization techniques into the system implementation.

QUEST gains the most benefits when evaluating the conjunctive predicates, since only the previous payloads can be delivered to upcoming predicate and QUEST can efficiently prune the most irrelevant instances based on the bitset-based predicates pushdown. But it can also efficiently process disjunctive queries based on the "or" bit computation. Because the "and" and "or" bit operations are exactly aligned to the conjunctive and disjunctive relations between predicates. We fell sorry to neglect this kind of situation and confused you. As for the nodes, they refer to the attributes in schema tree, and the operators refer to the attributes that are involved in a query, which is a subset of the nodes.

QUEST isn't developed based on an relational schema, but a novel unified nested tree model and pure columnar storage. Like other advanced databases designed for semi-structure nested records (*e.g.,* DREMEL[37], CORES[49], STEED[48]), QUEST pursues a efficient embedding scheme for nested document to avoid expensive cost caused by relational *JOIN* operator. Here we employed the natural simplification of CSR format (*i.e.,* Counter arrays) to encodes the one-to-many adjacent relations in nested tree. Thus, the query evaluation on nested tree is similar to graph traversal in graph database. However, we did not utilize the *EXPAND* operator used in most of graph databases (Neo4j .*etc*) to find pair of nodes that are connected through an edge, which is very similar to a relational *JOIN*operator. Because this operator will gradually expand a intermediate table as the nested path getting deeper which will incur unnecessary I/O cost and increase memory footprints. Thus, a bitset-based payload delivery and skip-tree based skipping scheme are developed to settle this problem. Specifically, the *SKIP_UP* and *SKIP_DOWN* operators have replaced the *EXPAND* operator as well as avoid the expensive relational *JOIN* operators when evaluating queries with deep nesting path, as such the intermediate results is maintained in a slight bitset form. There is an example for query evaluation in section 3.2.1 and corresponding illustration in figure 7. However, we will redesign more specific and appropriate examples to better describe the query evaluating process.

(5) What is $T_{indicator}$? Is it an ID or a pointer? Can it refer to a nested field or can it only refer to a record ?

Reply: Thanks for your careful reviewing, we are sorry the definition is not clear enough and confused you. $T_{indicator}$ is a type of node proposed in QUEST's extensive definition for nested records to express the many-to-one and many-to-many mapping relationships in graphs. Since QUEST's columnar storage layout is based on an order-preserving encoding, the ID can also perform as a pointer because it records the offset of the instance in data columns. It can refer to either a nested field or a specific data instance, since all of the data instances in a nested field are also stored in an order-preserving style in columns. There is basically no difference between the two forms.

O2. Unibench[50] has 10 queries but the experiments use 11 queries. Are you using Unibench?

Reply: We are sorry for misleading you due to our poor writing. We did not use Unibench, but take its choke points design and generation codes for reference to generate a new multi-model dataset with more complex data schema. And we further refine the choke points of scan-intensive cross-model analysis to design corresponding workloads referring to the choke points design of popular benchmarks such as LDBC [9] and TPC-H [14, 23].

O3. The main technique in the paper is the skip tree, which skips the nested levels in the schema trees. This technique is effective for nested document data. Then relational data and graph data are represented as nested trees. In this way, cross-model queries are supported. However, under the hood, the performance benefits come from the skip tree on nested document data.

Reply: Thanks for your careful reviewing and concise summary. Indeed, the skip-tree based skipping scheme is one of the most crucial designs of QUEST. In fact, it is constructed based on the unified logical representation and pure columnar physical data layout for multi-model data which is not trivial to achieve. As far as we know, there is still lack of such works that establish native unified columnar data layout to promote the efficiency of processing the scan-intensive cross-model analysis in academia.

(1) It may make sense to evaluate the performance impact of skip tree on JSON document data. Then the evaluation can use a large amount of real-world JSON document data.

Reply: This piece of advise is greatly helpful. We believe the skip-tree based skipping scheme could promote the efficiency of large scale of complex real-world JSON document data. Actually, we have done similar experiments on scan-intensive analytical workloads on a real-world medical JSON document from Pubmed to test the skip-tree's efficiency. QUEST outperforms other mainstream systems for nested documents like Parquet and CORES[49]. However, in this paper we mainly focus on extending such query evaluation scheme to the scenario of scan-intensive cross-model analysis which is a more timely and practical problem. Thus, we omitted the specific detailed experiments solely on JSON document data but viewed it as one situation of the cross-model analysis that the predicates' model distribution is slant as in Q10 where all of the predicates are performed on document. Thanks for your helpful advice and the detailed experiment results will be submitted in the following supplementary materials.

(2) The usefulness of the skip tree is highly dependent on the number of nested levels in the schema tree. If the schema tree is shallow, the benefits can be low.

Reply: Yes, you are right. If the nested depth is shallow, the benefits may be limited. In fact, once the nested depth is deeper than 4 (*i.e.,* we can skip at least one step), the skipping scheme can still ideally gains 20% – 30% efficiency promotion in query responding time due to the less I/O which is confirmed by our previous experiments contrasted to CORES on their nested dataset with 5 nesting depth that constructed from TPC-H by pre-joins. On the other hand, it also illustrates that QUEST is equipped with excellent robustness when the nested depth getting deeper which may be a challenging problem to other systems. Thanks for your helpful advice and the detailed experiment results will be submitted in the following supplementary materials.

(3) The skip ancestors and skip counters take extra space, which is not quantified.

Reply: Thanks for your careful reviewing. We are sorry for our poor writing that confused you. The size of skip tree index is

related to the number of specific data instance in each nested layer. Specifically, in our experiments, the space overhead is summarized in the first sentence in section 5.2.3: "QUEST takes 17.83 GB disk space which includes extra 2.85 GB space of Skip-tree index to store 20.24 GB multi-model data." The space overhead of skip-tree is acceptable. And the overall storage is much smaller than most of the competitors due to the excellent compression ability of columnar data layout.

(4) It would be nice if the evaluation compares Quest with Quest without skip tree.

Reply: Thanks for your helpful advice. We are sorry for the poor writing which buries the results of our skip-tree ablation experiments results in section 5.3.2 "We conduct more specific ablation experiment to evaluate the benefits of skip-tree structure in nested document-based data and graph-based data. The result shows that when we strip the skip-tree from QUEST to evaluate Q10 and Q11, the query running time increase 17

Minor: M1. one way to represent a graph is to create a vertex table, an edge table, a vertex property tableand an edge property table. The tables use integer keys as vertex lDs and edge lDs. This approach does notseem to cause huge storage redundancy' as suggested in Sec 2.1.

Reply: Thanks for your carefully reviewing, we are sorry for our inaccurate description. As you mentioned, there are other reasonable relational schema to store the graph. As far as we observed, in the edge table there are still some unnecessary IDs when different edges share the same source vertex. When there are a large number of super-vertex whose out-degree is large, the storage redundancy also can't be ignored. Thanks for your helpful advice, we will conduct more in-depth surveys and refine our wording.

TO REVIEWER #2:

O1. The scenario described seems quite artificial and oversimplified. Even the 'bad' solutions proposed to use a relational database as a single engine is not that bad; databases can handle quite efficiently joins across multiple tables and recursive queries involving nested records.

Reply: Thanks for your carefully reviewing. Indeed, as you mentioned, the relational databases can solve the problem in traditional way. However, the efficiency is hard to satisfy the need of modern data-driven applications when faced with intensive full-table scans and multiple many-to-many joins. As shown in our experiments, one of mainstream columnar relational databases Clickhouse, incurs the most often memory crash due to the huge intermediate results and sometimes causes huge query latency over 2000 seconds. Although the scenario is easy to understand and follow, it's hard to achieve satisfied performance using existing systems when evaluating scan-intensive cross-model analysis on large scale datasets in practice [34].

O2. The paper describes a method to transform various modalities into a relational schema using a columnar layout. This is a fine idea, but it raises a couple of concerns. The method is described in a very high-level, example-based manner. There is no evidence that this would scale in practical settings with various data models and at a varying scale. Assuming that this is a practical method, it needs to be presented in a sound, formal manner. For example, how one can guarantee that the method generates a lossless representation of record structure in a columnar format? How much nesting can be practically handled this way? Why is this the most efficient way to treat multi-modal data vs. for example querying fields of complex data types? Why even a relational engine is the right tool for this application? Several relational databases, including columnar ones, can handle quite efficiently nested records (such as JSON/Avro/... fields) without necessarily unflatten them; see for example Vertica's Flex tables or Redshift SUPER data type/PartiQL language, etc. It is not clear what is the novelty in the solution proposed -at least based on the examples described.

Reply: Gratefully thanks for your helpful advises. In the next version, we plan to optimize the writing style and manner of presentation to be more formal and rigorous. As for the first question, the correctness of pure columnar representation is guaranteed by the lossless representation of CSR format for adjacent matrix (i.e., the Counter and Indicator array) in graph, since the nested document and relation can be viewed as subsets of graph theoretically. And the extended definition for multi-model data is recursive, it can support arbitrary nested depth theoretically. QUEST's embedding scheme only focus on the mapping information between adjacent layers which is not directly related to nesting depth, rather maintaining the global structure information like Parquet [37] (i.e. repetition level and definition level for each instance). Thus, the QUEST's representation has good scalability for varying nesting depth and data amounts and excellent performance in practical applications. We can not theoretically prove it's the most efficient way, but QUEST did outperform other competitors in joint analysis across relational table, nested document and property graph. In fact, the superiority of columnar layout for analytical queries has been proven in both industry and academia. QUEST proposed a unified pure columnar data layout for multi-model data, significantly reducing the I/O cost and memory footprint when evaluating scan-intensive cross-model queries. As for the good scalability for more data models beyond relation, nested documents and graph, we take it as a future work to explore.

There is a little misunderstanding we want to clarify here. QUEST isn't developed based on an relational schema, but a novel unified nested tree model and pure columnar storage. Like other advanced databases designed for semi-structure nested records (*e.g.,* DREMEL[37], CORES[49], STEED[48]), QUEST pursues a efficient embedding scheme for nested document to avoid expensive cost caused by relational JOIN operator. Here we employed the natural simplification of CSR format (*i.e.,* Counter arrays) to encodes the one-to-many adjacent relations in nested tree. Thus, the query evaluation on nested tree is similar to graph traversal in graph database. However, we did not utilize the EXPAND operator used in most of graph databases (Neo4j *.etc*) to find pair of nodes that are connected through an edge, which is very similar to a relational JOIN operator. Because this operator will gradually expand a intermediate table as the nested path getting deeper which will incur unnecessary I/O cost and increase memory footprints. Thus, a bitset-based payload delivery and skip-tree based skipping scheme are developed to settle this problem. Specifically, the *SKIP_UP* and *SKIP_DOWN* operators have replaced the EXPAND operator as well as avoid the expensive relational JOIN operators when evaluating queries

with deep nesting path, as such the intermediate results is maintained in a slight bitset form. We have conducted similar research and experiments on several relational databases using the same multi-model dataset. The result shows all of them are faced with severe query latency and memory crash even the scale of dataset is not so giant. For more specific analysis, please refer to the section 5.2, in where ClickHouse is a mainstream columnar database based on relational schema which suffers the most often memory crash among all competitors. This is mainly because the restriction of structural relational schema and giant intermediate results caused by multiple many-to-many joins on tables. QUEST leverages pure columnar data layout and propose a more flexible and light weight query evaluation scheme rather than a relational manner, which is more native to multi-model data. Thanks for your helpful advise again, we are sincerely sorry for the misconceptions and misunderstanding caused by our poor writing. In next version of paper, we plan to rewrite the inappropriate examples and conduct more in-depth analysis in a more sound, formal manner.

O3. Similar concerns exist for the skip strategies. The precomputation step seems as overkill for real-world applications. The motivating example mentions a graph describing a social network; it is hard to see how such a precomputation step could be applied in such a case.

Reply: Thanks for your careful reviewing. Your concerns are very practical, we had the same concerns at the very beginning at this work. However, when conducted the contrast experiments with CORES[49] and Parquets[37], QUEST still gained nearly 20% – 30% performance promotion on a 5 layer nested dataset generated by pre-join TPC-H tables and a real-world medical document dataset from Pubmed with 7 nesting layers with slight increased storage overhead. And we find out that there are datasets with deeper nesting layers collected from industrial sensors. However, QUEST provides excellent robustness when the nesting layers getting deeper which may be a challenging problem to other systems.

We are sorry for our poor writing and presentation that confused you. We are sorry for being unable to illustrate all of the pictures and algorithms in paper due to the limited space. The precomputation on graphs is basically the same with the process of computing multi-hop adjacent matrix in graphs. Note that, the Counter and Indicator arrays can be viewed as a CSR(compressed sparse row) format of one-hop adjacent matrix. For example, there is a linked path A-B-C in graph schema where A,B,C are vertex label, we can compute the direct 2-hop adjacent matrix from A to C using two 1-hop adjacent matrices of A-B and B-C. Then the 2-hop adjacent matrix can be condensed into CSR form too, *i.e., skip-counter* and *skip-indicator*. The global precomputing strategy on graph also follows the skip-tree's initialization. Here is a more specific example shown in Figure 2

O4. How does the skip-tree based optimization method handles more complex query operators, besides simple filters? What class of filters (or other operators too) is supported? What kind of predicates/expressions can be covered? How do the search algorithms guarantee the right and/or optimal solution? A formal proof is also missing here. Determining an (optimal) filtering order based on picking each time the next most selective filter that haven't been explored that far is known greedy strategy in query optimization. In fact, more complex cases have been studied throughout the years

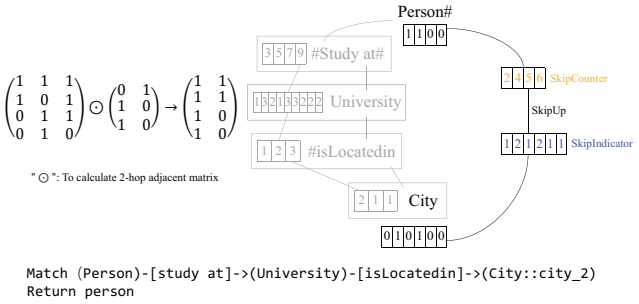

Figure 2: Illustration for precomputation on graph

in the context of query optimization (even as early as in System R, Volcano, Cascade, *etc.*). What makes this problem different from traditional query optimization?

Reply: Thanks for your careful reviewing, we are sorry for the poor writing that confused you. Since QUEST is mainly developed towards scan-intensive cross-model analysis, its optimized query evaluation components such as bitset-based query pushdown and cross-model payload delivery mainly focus on the conditional queries and cross-model join queries. However, these design principle does not conflict with other analytical queries like aggregation, sorting and grouping, etc. As mentioned in CORES[49], these operator processing cost on top of highly selective filters is relatively much smaller than the I/O cost caused by data scans in disk-based storage systems, thus we omitted the specific analysis in paper. However, the proposed query evaluation scheme can be seamlessly used to optimize the processing of scan and join operators and to better support other types of analytical queries. We are sorry for our negligence and poor writing. The formal proof will be presented in next version of the paper. Basically, the filtering information maintained in bitsets will only be lost during the skip up process, *i.e.,* from many descendants to one ancestor. Because as well as there exist one descendant that is true for the filter i.e. its bit is set to 1 , the bit of the ancestor will also be set to 1. If the filter path back to the descendants again, the ancestor's bit 1 will be copied to all of its descendants, and mask previous filtering results on descendants. Thus, a post order traversal on tree is needed here. Indeed, there have been lots of studies on the relational query optimization throughout these years. However, as we mentioned before, our query evaluation scheme is not based on a relational execution engine, but a nested tree traversal in a semi-structure manner. And QUEST stores multi-model data in a pure columnar format, and achieve fine-grained query push-down which makes our cost function different from traditional relational query optimization. Because we need to take the length of tree traversal (every step will incur additional I/O of retrieving metadata) and selectivity of predicates into account to formalize the comprehensive optimization problems. Although it may seem like a classic greedy strategy, QUEST's cost model tightly combines the structural characteristics of multi-model data and unified column layout and is different from traditional optimization strategies.

O5. The experimental analysis is very limited.

(1) There is no description of the implementation and the maturity level of Quest. The datasets used are quite small, and although one appreciated the resource constraints, investigating simple queries in such a small setting is not ideal to evaluate how does this approach scale. Several significant overheads (*e.g.,* precomputation step) might not appear to a full extent with so small data (*e.g.,* a single relation table with a storage footprint at 6MB).

Reply: Thanks for your careful reviewing, we are sorry for our negligence. We plan to refining the detailed description of experiment implementation. We employ the AVRO serialization framework and optimize the CORES's open source repository to achieve Counter and Indicator based embedding scheme and unified columnar storage data layout. And then the precomputation is implemented to construct skip-tree index based on it . As for the size of datasets, we generate our multi-model datasets based on the partial LDBC SNC graph dataset scaled in SF30. A nested document and relational table dataset is generated, where the number of Person instance is nearly aligned to the number of user instance in graph data. But the relational schema is not complex, which makes its storage foot print too small. However, similar situation appears in Unibench[50] which is one of the most popular multi-model database benchmark, where the size of relational table is 15.8 megabytes, size of graph is 6191.5 megabytes and size of JSON document is 6184.9 megabytes at its largest datasets' scale. A number of experiments for testing multi-model and poly/multi-store are conducted on Unibench [30, 45, 50]. In fact, the multi-model datasets used in QUEST's experiment is relatively larger than the largest data scale in Unibench. As for the size of skip tree index, it is related to the number of specific data instance in each nested layer. Specifically, in our experiments, the space overhead is summarized in the first sentence in section 5.2.3: "QUEST takes 17.83 GB disk space which includes extra 2.85 GB space of Skip-tree index to store 20.24 GB multi-model data." The space overhead of skip-tree is acceptable and the overall storage is much smaller than most of the competitors due to the excellent compression ability of columnar data layout. But as you mentioned, our experiments are limited, we will take your helpful advise and conduct more specific experiment on larger datasets to test QUEST's scalability in data amount on more high-end severs.

(2) Surprisingly, a relational/columnar database is not evaluated in the experiments. This could be particularly interesting in terms of measuring query performance, disk storage, etc. which are presumably the sweetspots of such systems. Similarly, a comparison with multistore/polystore system would also be revealing.

Reply: We are sorry for our poor writing and confused you. In fact, we have conducted the contrast experiments with Clickhouse, a popular open source columnar relational database. Indeed, it outperforms the two multi-model databases in most queries, but faced with larger memory footprints and the most often memory crashes due to the huge intermediate results by multiple joins. However, QUEST still gains more than double performance promotion compared to Clickhouse in most of the queries and with less memory usage. As for disk storage, Clickhouse did take the least space to store the overall multi-model dataset. This is gained from its superior compression techniques on pure columnar data layout for each table which is not the focus of QUEST. However, we are working on similar techniques that evaluating queries directly on the condense compressed structure to further reduce disk storage and memory footprints. As for the comparison with multistore/polystore systems, there is detailed experimental analysis [45] to test the pros and cons of mainstream multi-model databases with polystores. The results shows the polyglot persistence setup yields a much worse execution time when evaluating complex queries that combine data from three distinct data models (respectively a factor 60.2 and 57.9 increase in execution time when comparing the mean values to the ArangoDB mean result). This is mainly because queries are performed in three distinct databases and the result sets of these queries are then processed programatically in the polyglot implementation. In these examples, this proves to be significantly more sub-optimal than the performance of a single cross-model query. In fact, in our experiment, QUEST improves the performance by 3.78x to more than 100x (ArangoDB triggered query timeout in our setups) when evaluating the queries that combine the three types of model. The performance promotion mainly gains from the unified columnar data layout and bitset-based payload delivery. However, we will supply more important experiments to test the pro and cons of QUEST more comprehensively.

(3) The comparison of Quest with the other systems is a bit apples-to-orange. It is not clear how the times reported correspond to the various overheads of each system/method and whether the comparison measures similar functionality or in some systems may include additional overheads/startup costs that do not relate with the measured query behavior. Also, does each system run the same 'query plan'?

Reply: Thanks for your carefully reviewing, we are sorry for our poor writing and confused you. We tried our best to ensure the fairness of the experiment setup including the same hardware configuration, independent virtual environment, and data warm-up for each database before performance testing to reach a stable state to ensure there is not additional overheads that do not relate with the query behavior. In all experiment, we use default indexes which are built on primary keys, and no secondary index is created. And for each query, we calculate the average 10 experimental results as the final results. However, due to the heterogeneity of different databases which may have different types of model as their first citizen (e.g., document, graph, table) and different operators(e.g., JOIN in relational database, EXPAND in graph database), it's hard to guarantee they use the same query plan. But we should further improve our experiments to conduct more in-depth analysis including the pros and cons of different operators used in different types of databases and to further control the variables in experiment. Sincerely thanks for your helpful advise.

TO REVIEWER #3:

O1. Using the an adjacency list to model graphs is pretty standard. Most graph databases use the same model already and the nested data can be viewed as acyclic graphs while existing nested data formats (e.g., Parquet) use similar models to store the structure. It is not quite clear why the authors stress the multi-model aspect of this as both the relational and nested models are subsets of the graph model in this idea. The authors should clarify what distinguishes their system from a graph database that uses adjacency lists and can also store nested data as an acyclic graph.

Reply: Thanks for your carefully reviewing, your comments are very insightful. Indeed, using an CSR format of adjacent matrix to store the mapping information in graphs is pretty standard. As far as we know, there is a lack of studies to leverage this classic structure to achieve unified columnar data layout towards scan-intensive cross-model analysis which is a practical and timely problem. During the research, we find out that the CSR format for one-to-many mapping (*i.e.,* Counter array in CORES[49]) can also be viewed as a efficient embedding scheme for nested document which is slighter and more flexible than the Parquet's global embedding scheme(*i.e.,* Repetition level and Definition level for each instance). Because CSR only focus on the mapping information between adjacent layers, cooperated with the bitset transfer for intermediate results delivery, it can support arbitrary query paths in nested schema. Thus, the query evaluation on nested document can be viewed as tree traversal in a more efficient way rather than the flattening techniques developed by Parquet [5, 37]. And we find out that the I/O cost and memory footprints could be further reduced by precomputing the mapping information between ancestors and descendants with little increased storage overhead, thus the skip-tree based skipping scheme is developed to accelerate the bitset transfer. In fact, if one directly utilizes a graph database (*e.g.,* Neo4j) to store nested documents without any optimization, several concerns on query processing efficiency are raised. First, in many graph systems, *EXPAND* operator is used to find pair of nodes that are connected through an edge which is very similar to a relational *JOIN* operator. This operator will gradually expand a intermediate table as the nested path getting deeper which will incur unnecessary I/O cost and increase memory footprints. Take the nested tree in Figure 5 in QUEST's paper as an example, retrieving the data instances in node 17 that are connected to instances in node 14, the graph engine will invoke *EXPAND* operator 10 times and generate a giant intermediate table when the data scale is large as a 10-hop traversal is needed in graph here. These concerns are also reflected by our experiment results, as the Neo4j suffers the largest memory footprint in most of queries and longer query responding time than QUEST and CORES on pure document data. And as shown in table 3, the storage overhead is relatively larger since some unnecessary indexes will take ultra space. QUEST's bitset-based payload delivery and skip-tree based skipping scheme significantly reduce the size of intermediate result and promote the efficiency of tree traversal. It replaced the EXPAND operator with a two pair-wise operators *SKIP_UP* and *SKIP_DOWN*. And it can also be efficiently applied to graph data based on graph's spanning tree. Thus, the overall designs of QUEST reconcile the characteristics of both nested document and graph data to achieve the excellent performance for evaluating scan-intensive cross-model analytical queries.

O2. I did not find a stand-alone comparison of what the benefit of the skip-tree is in the system. I would expect the authors to include a comparison with stand-alone nested scanners such as Parquet to show the strength of this technique on scanning nested data. The authors already include comparisons with graph databases but the paper should include the impact of the skip-tree.

Reply: Thanks for your helpful advice. We are sorry for the poor writing and presentation which buries the results of our skip-tree ablation experiments results in section 5.3.2 "We conduct more specific ablation experiment to evaluate the benefits of skip-tree structure in nested document-based data and graph-based data. The result shows that when we strip the skip-tree from QUEST to evaluate Q10 and Q11, the query running time increase 17% and 16% as well as the memory usage increase 5% and 2%.". The results shows that the skip-tree has great affect on the QUEST performance in both responds time and memory footprint. And it's obvious that as the nested depth gets deeper, the absolute I/O reduction gains from skip-tree will become even more prominent. Actually, we have done similar experiments on scan-intensive analytical workloads on a real-world medical JSON document from Pubmed to test the skip-tree's efficiency. QUEST outperformed other mainstream systems for nested documents like Parquet[37] and CORES[49]. However, in this paper we mainly focus on extend such query evaluation scheme to the scenario of scan-intensive cross-model analysis which is a more timely and practical problem. Thus, we omitted the specific detailed experiments on JSON document data but viewed it as one situation of the cross-model analysis that the predicates' model distribution is slant as in Q10 where all of the predicates are performed on document data.

O3. The part of the paper that discusses the query optimization model is very generic. Basically I would identify two questions as the most relevant:

1) How to determine the order at which to store the data in a graph schema (which node is used as the root). Nested and relational data are straight-forward but the graph data have multiple possible representations.

Reply: Thanks for your carefully reviewing. This is a very interesting problem. In fact, this problem also plagued us at the very beginning of this work. Our suggestion is to choose the vertex with the highest out-degree as the root in graph schema (e.g. Person in LDBS SNC). In the implementation of QUEST, we use a cardinality-aware depth first search to traverse the vertex with smaller cardinality first. However, any appropriate traverse order from a possible root can results in a expanding tree of graph. We conduct experiments to test the influence of different expanding scheme on same queries, there is not obvious performance gap between any two expanding scheme. But it is still a insightful problem that worth to be further explored.

2) How to estimate the cardinality of each predicate in order to minimize the intermediate results and optimize the rest of the plan (e.g., join order). The authors focus on not traversing the same sub-tree twice but this too obvious, the more interesting question is how do you sample the nested/graph data to determine predicate ordering assuming the data is stored in this layout.

Reply: Thanks for your careful reviewing. This is a very interesting problem! The suggestion is highly commendable and worth more in-depth discussion. After careful consideration, we opted for enumeration due to the limited input size (The nested depth is normally less than 20), which proved to be efficient in our experimental analysis. Consequently, we did not delve further into it; however, from a theoretical algorithmic perspective, it remains an excellent point. The (parametric) complexity of this optimization problem will be further explored in the context of unbounded input size, along with corresponding optimization algorithms, parametric algorithms, approximation algorithms, and so forth. In the current implementation, the predicate ordering is realized in a manual-act

manner. Although the cardinality estimation is not the main focus of QUEST for now, we are integrating the relevant advanced techniques into our system and trying to develop more appropriate techniques based on QUEST's data layout for cross-model analysis.

REFERENCES