Cephalopod – Virtual Data Model Composition through Partial Query Translation

Holger Pirk
hlgr@imperial.ac.uk
Imperial College London
London, UK

David Conor Loughlin
david.loughlin19@imperial.ac.uk
Imperial College London
London, UK

ABSTRACT

Most applications have an ideal data model they should be supported by: business data by relations, social networks by graphs, messaging applications by documents and machine learning by vectors. Unfortunately, many applications need to be implemented against a "less-than-ideal" (we use the term "imposed") data model: business data is stored in documents, learned models must process relational tuples and graphs are embedded in vectors. The textbook solution to that problem is physical integration: Extracting, Transforming and Loading data from the imposed into the ideal data model. While effective, this ETL-process is expensive and leads to staleness. Virtual integration (through query rewriting) avoids these problems but leads to a combinatorial explosion of ideal-to-imposed-model mappings.

We propose to address this problem by developing a "Bridge Representation" that can be used to implement virtual integration through query translation when possible and physical integration through data transformation when necessary.

In this paper, we outline the idea, study a number of guiding use cases and develop a research agenda towards such a Bridge Representation and a system that implements the approach. We also provide some preliminary results indicating that even non-bijective data-model integrations like vector embeddings can be supported at a fraction of the cost of physical integration.

1 MOTIVATION

Most applications are built on top of a logical data model that provides functionality such as storage of *data-primitives* like tuples, graphs or vectors, *operators* like joins, edge-traversal, or vector-products and even the enforcement of *integrity constraints* like dimensionalities, uniqueness or connectedness. Selecting an appropriate data model for an application is a key design decision that affects developer-productivity, performance and quality of an application throughout its lifecycle.

Consider, e.g., the case of Retrieval-Augmented Generation (RAG), a technique that has recently been proposed to address two of the key shortcomings of Large-Language Models (LLMs): the expensive model refinement when new data comes in and the model's tendency to "hallucinate" facts when generating responses. Under RAG, an LLM receives a set of relevant input data just before a prompt is submitted. If that data fits into the "context window" of the model and is relevant to the question, RAG significantly improves result quality, reduces hallucinations and removes the need for retraining [Lewis et al.(2020)]. To determine the most relevant data for a prompt, RAG performs a top-k similarity search on vector embeddings of all data items. This, naturally, requires data to be stored as Vectors in an appropriate Data Management System (DMS). Unfortunately, most data is not stored in vector databases and has to be embedded before being used for RAG. This embedding is expensive: given, e.g., the current OpenAI-pricing, the cost of embedding all global non-spam email traffic (361 billion emails of, on average, 434 words) would exceed $20 million per day.

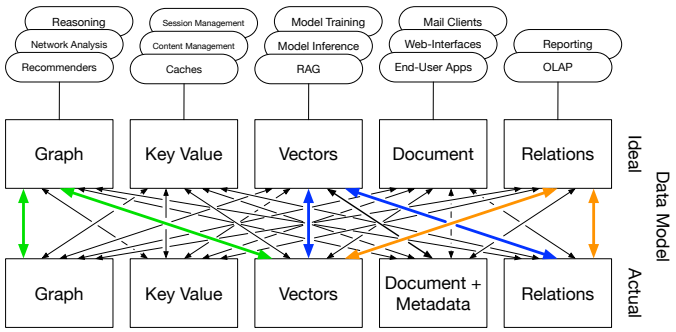

Figure 1: Applications, Appropriate & Imposed Data Models

RAG is by no means the only application in which "appropriate" and "imposed" data models diverge: the need to make newly developed software interact with existing infrastructure, combine data from different sources or accommodate the adoption of a specific technology in an organization often imposes a data model. Arguably, any combination of appropriate and imposed data models arises in practice, leading to a landscape like the one illustrated in Figure 1: different applications (the top row) call for different appropriate data models (the middle row) but must be supported on any of the existing models (the bottom row). The state of the art to bridge the gap between imposed and appropriate models is Extract, Transform & Load (ETL): in regular intervals, data in the imposed model is extracted from the source, transformed into the appropriate model and loaded into a system designed for the appropriate model. ETL requires the development and maintenance of ETL-"pipelines", is costly to execute and renders data in the appropriate model stale when new data arrives at the source.

We propose *to address all of these issues by replacing costly and eager ETL by on-demand translation of queries against one data model*

 15th Annual Conference on Innovative Data Systems Research (CIDR '25). January 19-22, Amsterdam, The Netherlands

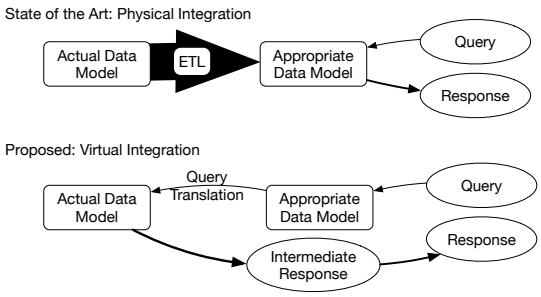

Figure 2: Physical vs. Virtual Data Integration

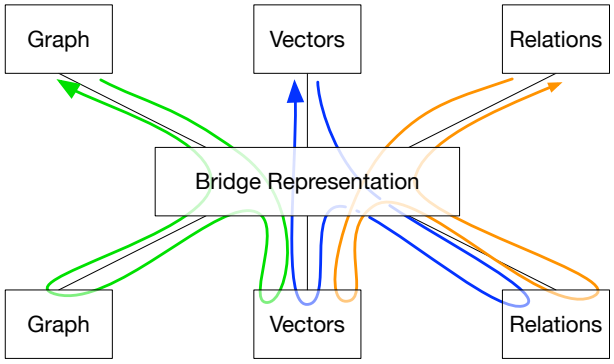

Figure 3: Virtual Data Model Integration

into queries against another model. Figure 2 contrasts the state of the art with the approach we propose.

1.1 Key Idea

Data model integration is a long-standing challenge in data management. Due to its complexity, it has traditionally been solved by transforming data from source to target model. Building on the transformation as a formal definition of the mapping between source and target model, *we propose to a) invert the transformation function, b) apply it to the query rather than the data and c) generate plans that opportunistically combine data-transformation and query-translation to achieve (close-to) optimal performance.*

While there have been some recent advances in the development of data-oriented Intermediate Representations (IRs) for query execution, current IRs aim to be general enough to capture arbitrary execution, serve to generate code for any platform (CPUs, GPUs, FPGAs, TPUs, etc.) and allow any of the optimizations that is supported by modern compilers. This broad ambition led to many competing standards, dialects and frameworks, with none of them being close to achieving universal adoption (LLVM IR and MLIR being, at least, close for the sole purpose of executable code generation). This broad ambition makes them unfit for the translation of queries between high-level query languages and data models.

Instead, we propose an approach that specifically aims at data-model integration: inspired by the idea of "Bridge Languages" in linguistics, we aim to develop a **"Bridge Representation"**, i.e., an IR with two objectives: first, to translate the parts of query plans from the appropriate to the imposed when that is possible and beneficial and, second, to efficiently transform data from imposed to appropriate data models when query translation is not possible or beneficial. To maximize performance, the decision to "**translate (queries) or transform (data)**" shall be possible either at the granularity of the entire query plan or parts of it. We illustrate the concept in Figure 3 and refer to it as "**Partial Query Translation (PQT)**" throughout this paper: original queries (implemented against one model) are partially *translated* into queries against another, those parts are evaluated, the results (and the rest of the query) *transferred* to another model, the rest of the query evaluated and the result returned to the user.

2 GUIDING USE CASES

The idea of virtual data model integration through Partial Translation (PT) can be applied to any combination of appropriate and imposed data models illustrated in Figure 1. However, we propose to guide our efforts by focusing on three high-impact use cases chosen to cover a large part of the problem space while keeping engineering effort moderate. The flow of each of these cases corresponds to one of the colored query/data-flows illustrated in Figure 3.

Blue Case: Retrieval-Augmented Generation on Unembedded Data. In the context of the RAG case, we outlined in Section 1, the ground truth of an organization's data usually lives in data sources that do not expose vectors as their data model: email servers, word documents and relational databases. These do not support vector embeddings or operations on them. They do, however, support the filtering and sorting of data using per-data-item predicates. We propose to generate filter predicates to extract and embed only data items that are likely to be relevant, thereby reducing cost by many orders of magnitude.

Green Case: ML-Accelerated Reasoning on Knowledge Graphs. In the context of "Classic" Artificial Intelligence, reasoning on Knowledge Graphs is the de-facto gold standard: it is precise, explainable and free from false positives. However, it is computationally expensive. Fact prediction based on Graph-Neural Networks (GNNs) is faster but prone to produce false positives (as well as, less frequently, false negatives) [Zhang and Chen(2018)]. A hybrid approach has the potential to combine the strengths of either approach: the GNN can quickly predict facts, and the classic reasoner can confirm or contradict them. As illustrated in Figure 3, we propose to support this case by partially processing each query in a vector-processing engine before sending it to a graph-processing engine.

Orange Case: Supporting Vector-Operations in Relational Data Management Systems. The last driving use case relates to classic relational data processing: cloud data processing system vendors like Microsoft [sql([n. d.])] and Snowflake [sno([n. d.])] have recently extended their offerings with Vector-operations to support operations like similarity search and image-processing. However, to minimize the impact on existing systems, vendors built on existing unstructured datatypes, i.e., JSON, String/Varchar or Binary/BLOB.

However, retrofitting vector-operators onto a relational system induces overhead, complicates use and obfuscates optimization opportunities. Offloading vector operations to a purpose-built vector database kernel or system will combine the best performance of either system. Partial Translation and Transfer through a Bridge Representation Representation is the key enabler for this.

3 CHALLENGES

The key problem we aim to reduce is the high cost of data transformation when serving data in one model in another model. We can break this problem down into three challenges.

Partial Query Translation

The first challenge in supporting any of the outlined cases is determining how an operation on one data model can be translated into another. Some operations, such as single-step graph traversals and relational joins, are known to be equivalent, and bijective translation functions are well-established [Paradies et al.(2015)]. Inverting a data transformation to create a query translation is, therefore, tractable. Others, like vector embedding, however, are not invertible and may even be lossy, which prevents accurate query translation. In Section 7, we provide preliminary results that suggest that many of these can, however, be over-approximated. This will significantly reduce the size of the dataset that requires transformation. After the transformation, the set can be refined to eliminate false positives.

To meet this challenge, both bijective as well as injective transformations need to be supported, which raises **research questions** such as *"What languages/queries can be fully translated?"*, *"Are there characteristics of data models that cannot be replicated in other data models?"* and *How can injective data transformations be inverted through over-approximated query translation?*. In particular, the last one is intellectually exciting.

The Bridge Representation

The second challenge is to avoid the combinatorial number of required model-to-model transformations by developing an appropriate Bridge Representation (BR) (as illustrated in Figure 3). While similar to "classic" data-oriented IRs [Palkar et al.(2017), Pirk et al.(2016), Lattner et al.(2021), Funke et al.(2020)], the required Bridge Representation has fundamentally different requirements: where classic IRs are **intended as steps towards the generation of low-level/executable code**, we propose to develop a representation designed to (opportunistically) translate one high-level language into another.

Such a Bridge Representation inherits some of the requirements of classic IRs: it should be succinct, machine-readable and small with respect to the number of exposed primitives. A BR, however, goes beyond the requirements of an IR: it must occupy a different point in the design space of query representations.

To illustrate this point, consider Figure 4. It displays three desiderata of query representations: first (on the x-axis), it should be general enough to represent any query on any data model. Second (on the y-axis), it should capture semantic information (formal operator semantics, equivalence, statefulness, etc.). Third (on the z-axis), it should contain the information to re-constitute the original query (or at least an equivalent query) from the representation. We term this property "Fidelity". To the best of our knowledge, all query representations strike a compromise between these desiderata: Plain Text, e.g., is highly general and has high fidelity but offers no semantic information. Algebraic plans provide semantic information at the expense of generality. Classic IRs offer generality and semantic information but make it exceedingly hard to reconstitute a high-level representation. Given the purpose of each of these representations, the trade-offs are well-justified. A BR, however, requires all of these desiderata, making its development challenging.

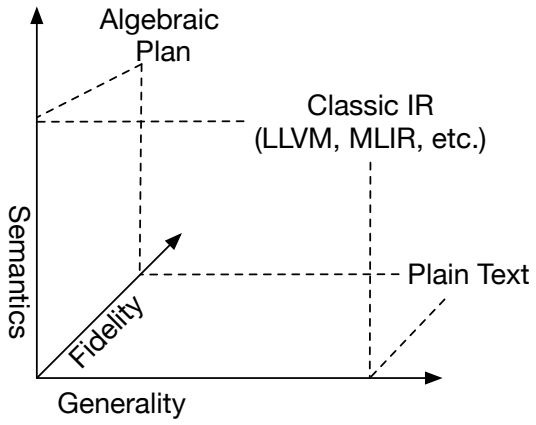

Figure 4: The Query Representations Design Space

However, some properties of IRs are not required for BRs: the need to be optimizable is strongly relaxed as queries will be optimized in the target system. Further, generality is a "soft requirement": if a specific operation is fundamentally not representable in the BR, it can be represented as a "gray box", i.e., in a form that can be "decompiled" into the language it was generated from but not translated into any other language. Such gray-box operators require data transformation at runtime, they are, in some cases, necessary to faithfully capture a data model. Note that while plain text would be an appropriate representation for the purpose of decompilation (due to its high fidelity and generality), it fails to capture any semantics, preventing effective translation to other data models.

Research questions associated with this challenge are *"How can queries be represented, translated and rewritten using a Bridge Representation?"* and *"How is the semantic relationship/mapping of operators in different systems best represented to simplify the translation/transformation process?"*.

Optimizing "Transfer vs. Translate"

While addressing the challenges outlined above enables the translation of queries, query translation is not always preferable to data-transformation: depending on factors like operator implementation, indices, execution model or optimizer decisions, transferring data to

a more efficient system can be beneficial. Given the focus on cross-data-model querying, the decision to translate or transform should be made cost-based and at subplan-granularity. The cost model must not only reflect "static" properties of each system (operators, optimizer and execution model) but also "dynamic" properties of the database (indices, memoization of transformed data or the need to invalidate transformed data upon update).

This requires answering research questions such as *"What heuristics and/or cost models can be used to decide on which system to evaluate what operator?"*, *"How can systems transfer data efficiently and securely when that is necessary (e.g., by defining external tables or using shared memory)?"* or *"How can data be effectively memoized, updated and invalidated when that is necessary for freshness or privacy reasons?"*.

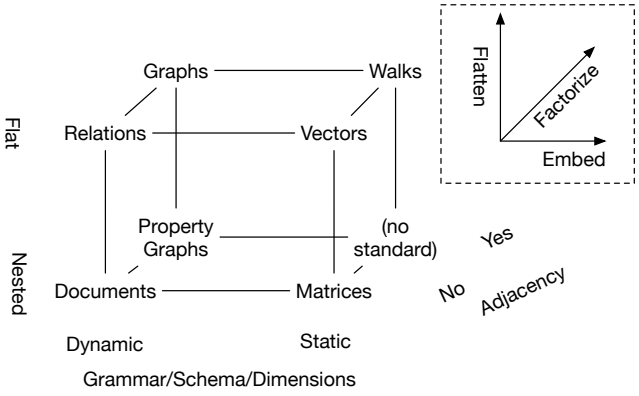

Figure 5: Dimensions of Structured Data Models

4 APPROACH

Building on BOSS

My group at Imperial College has, in the last three years, developed *BOSS*, a data management system pioneering a new class of systems: designed from scratch to allow easy, overhead-free composition from specialized components. We have demonstrated the utility of this architecture to manage hardware-heterogeneity [Mohr-Daurat et al.(2023a)], support machine learning workloads [Mohr-Daurat and Pirk(2021)] and impute missing data [Mohr-Daurat et al.(2023b)]. We are currently working on the integration of storage/compute-disaggregation and adaptive query processing strategies to exploit patterns in the data.

BOSS is built on two design principles: the first is the ability to represent partially evaluated queries by allowing the free composition of unevaluated operators (i.e., code) and results of operator evaluation (i.e., data) in a single, unified representation. The second is the ability to move data between components without the need for costly data copies through the use of a technique we call "destructive decomposition and recomposition". These two principles make BOSS the ideal platform for the composition of data models we envision. We have already started to explore and implement the presented ideas as extensions to BOSS.

Directions

Based on the BOSS DBMS, we envision work in five directions.

BOSS Backends. Restricting our focus on relations, graphs and vectors, we propose to integrate kernels supporting these data models into BOSS. For relations, we already integrated Meta's Velox [Pedreira et al.(2022), Mohr-Daurat et al.(2023a)], for vectors, we have started to integrate the Milvus kernel [Wang et al.(2021a)] and, for graphs, the Boost Graph Library [Siek et al.(2001)]. All of these will be integrated into BOSS as "Engines", i.e., libraries conforming to the existing BOSS APIs. While integrating existing libraries as Engines requires the development of wrapper code, we do not foresee substantial challenges in that effort.

Bridge Representation. The heart of this work is the Bridge Representation to connect the imposed and appropriate data models. Building on the existing BOSS query representation, the BR will be implemented in the form of a grammar for symbolic expressions. The grammar will, to the extent possible, reflect the commonality between operators of the different data models but also allow data-model-specific extensions to that "common core" (which can, then, be grounded in rewriting rules).

Rewriting. This effort will build on the BR, complementing it with a definition of equivalences of the data-model-specific/non-common operators. Capturing equivalence of operators of different data models, while non-trivial, is still possible through careful analysis of the solution space.

Figure 5 illustrates that space and notable points in it along three dimensions: data model flat- vs. nested-ness (y-axis), dynamic vs. static dimensionality (x-axis) and the existence of an (inherent) notion of adjacency (i.e., an implicit *neighbour* relationship) on the z-axis. Interestingly, data models can be located in that space fairly accurately. The figure also illustrates the (well-known) techniques to transform data in one representation into data of another (note that often, multiple different instances of a transformation are known). The *flattening* [Ulrich and Grust(2015)] and *factorization* [Olteanu and Schleich(2016)] transformations are invertible (through *unflattening* and *multiplication*). The bijective nature of the transformation allows the translation of queries and data along those axes. The *Embedding* transformation, however, is generally not invertible, preventing the accurate rewriting of a vector-query into a relational-, graph- or document-query.

However, our results in Section 7 suggests that a function that determines filter predicates from an embedding transformation can be learned by a neural network through self-supervised learning. While the learned function is likely to yield false positives, these do not constitute a result quality problem as they can be removed when generating the final response. They do, however, increase the cost and response latency, e.g., in RAG-applications as false positives will require extraction, embedding and ranking. By tuning the filter-generating network to increase recall at the expense of lowered precision, users can tune the system to achieve the performance-to-quality tradeoff their application requires.

To implement partial query translation among data models, we plan to follow state-of-the-art Multi-Level IRs (like MLIR): these define equivalences in the form of "virtual operators", i.e., operators that do not necessarily have an implementation but can be reduced to other, data-model-specific operators. Once those operator-equivalences are established, model-specific query plans can be rewritten using state-of-the-art plan transformation (i.e., rule-based rewriting) and "answering-queries-using-views" [Halevy(2001)] techniques. To implement those, we will build on BOSS' support for partial query evaluation to develop a rewriting framework for BOSS.

Optimization. This direction will build on the rewriting framework and the availability of internal cost estimates of data-model-specific systems as well as offline training to allow the estimation of cross-kernel-processing costs. By capturing (internal) processing cost with learned transformation/transfer cost, we plan to develop a holistic optimizer that minimizes overall execution cost.

Due to its low cross-kernel migration costs, BOSS is the ideal platform to compose data-model-specific kernels. BOSS is designed to maximize extensibility and allows the integration of kernels with widely varying semantics, rewrite modules that opportunistically translate queries between them and (cost-based) optimizers. It even allows the "injection" of custom code for data-model conversion as a User-Defined Function if a kernel supports that. The key enabling techniques for overhead-free composition are the avoidance of data copies through a novel cross-kernel, single-owner/move-centric memory-management model and the ability to store operators and intermediate results in a query plan.

Instances. In the final direction, we plan to develop applications supporting each of the guiding use cases. These will serve as validation for the concept but also solve practical data-science problems. For the orange case, we will develop a hybrid vector/relational system behind a standard JDBC/ODBC driver. It will provide the illusion of vectors as native datatypes in any relational DBMS. For the blue case, we will develop a system to make business-data available for querying through LLMs. For the green case, we will implement a GNN-accelerated graph-reasoning framework (with a SPARQL-like query API).

5 PRELIMINARY RESULTS

To indicate the viability of the approach, we conducted a preliminary study using email data. We sampled 16000 emails of a single recipient in the Enron email corpus [William W. Cohen(2015)], and calculated a two-dimensional vector embedding of the email bodies using the "thenlper/gte-small" model [Li et al.(2023)] (available via HuggingFace)[1].

Figure 6 plots the 2-D embedding of the email body on x and y and encodes the sender in the color of the points. The top-4 senders are encoded in unique colors, while all other senders are purple points. We observe a strong correlation between embedding and

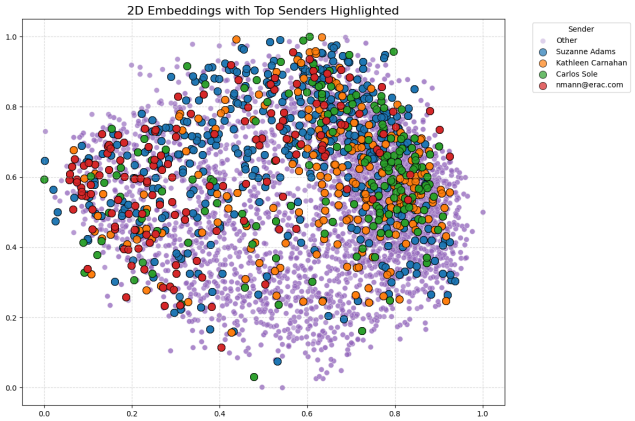

Figure 6: Correlation of 2-D Email Body Embeddings (X- and Y-axes) and Senders (Color of the Points)

sender: specifically, we observe that most green points cluster in the top right, while red points cluster on the left and top middle.

These results support the intuition that vector embeddings of the body (which are designed to capture the textual content) correlate with the sender – most emails originating from a person cover similar content. To explore opportunities for exploiting that correlation to accelerate RAG applications, we experimented using the sampled recipient's email data from the Enron corpus, in CSV format, and compared the execution of vector-similarity queries performed with and without pre-filtering on the sender.

To perform this experiment, we implemented a system using several BOSS engines forming an execution pipeline. Concerning the queries, we randomly sample 5 unique emails from the data set, calculate the embeddings of their bodies, and execute a top-10 vector-similarity search for each embedding.

We execute queries with pre-filtering in a system comprised of 5 BOSS engines, the first is the CSV Loader Engine responsible for loading the CSV email data into a BOSS table. The second engine is the Sender Prediction Engine which predicts the most likely sender for the query embedding using a simple neural network we pre-trained. The third engine, called the Relational Volcano Processing Engine, filters the input data on the previously predicted sender. After pre-filtering, the Embedding Engine (fourth engine) uses the "thenlper/gte-small" model to calculate a 384-dimensional vector embedding for each remaining body. Finally, the Vector Database Engine (the fifth engine) performs the similarity search using the execution engine of the milvus vector database (Knowhere) [Wang et al.(2021b)]. When executing the queries without any pre-filtering, the system is instead composed of the CSV Loader, Embedding, and Vector Database engines.

For hardware, we performed the experiment on a server with two Intel Xeon Silver 4114 2.20 GHz CPUs, each with 10 physical cores, a 14 MB LLC cache and 196 GB of memory. To run the experiments we use Ubuntu 18.04 with Linux Kernel 4.15.0-209 and compile all code with Clang version 14 using compiler flags -O2.

[1]Note that the low dimensionality is for visualization purposes. In practice, vector embeddings are between 32 and 4096 or more depending on the data and application.

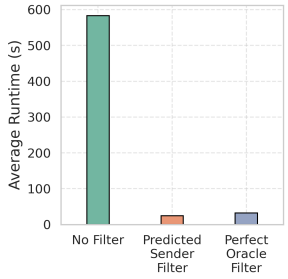

Figure 7: Average Runtime for Calculating the Embeddings of Email Bodies

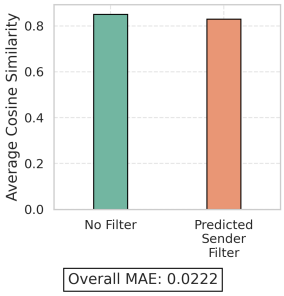

Figure 8: Average Cosine Similarities Scores for Top-10 Similarity Search

Figure 7 plots the average runtime to embed the email bodies in both the non-pre-filtering and pre-filtering approaches. We also plot an additional pre-filtering approach in which the sender prediction model is replaced with an oracle that returns exactly the senders of the emails in the query results produced by the non-pre-filtering approach. The plot shows that the embedding time for the entire dataset is orders of magnitude greater than that of either the predicted sender or oracle approaches. This is due to the pre-filtering approaches reducing the number of bodies for which embeddings are calculated from 16000 to between 200 and 1000.

These embedding runtime results indicate significant performance improvement, but this performance may come at the expense of quality. Figure 8 plots the cosine similarities, averaged across the results, for both the non-filtering and predicted sender pre-filtering approaches, and it shows the mean absolute error (MAE) between the two approaches. We consider the shown cosine similarities of each approach to be close with a small MAE. This indicates that, while the sender prediction model does predict the senders perfectly, the pre-filtering still returns email bodies very similar to the query email body.

Together, Figure 7 and Figure 8 show that pre-filtering using metadata, like the sender of an email, can offer great performance improvement without sacrificing much of the quality of the results of top-k similarity queries. Naturally, the simple sender prediction model we used could be improved upon using more sophisticated architectures and more training data to offer even higher-quality results. Further, pre-filtering is not limited to a single attribute, indicating there are more opportunities to reduce the set of candidates that need to be transformed.

6 RELATED WORK

In its effort to combine different data models, our work is most obviously related to the work on *Polystores* [Duggan et al.(2015)]. However, we innovate in several respects: first, we provide a foundational underpinning to Polystores by capturing the relationship between different models, where systems like BigDawg rely on ad-hoc shims to connect data sources (without providing guidance on how these shims are implemented). Second, we aim to provide not only location transparency but semantic transparency as well – users will experience the illusion of querying data in the appropriate data model even if the data is stored in a different one. Last, we aim to provide tighter integration, ideally at the level of plans and operators rather than declarative queries.

In partially evaluating subqueries, our work is related to ideas of answering queries using views [Halevy(2001)] and Query Decomposition [Mackinnon et al.(1998)]. However these approaches focus on schema-heterogeneity but assume a common data model. In contrast, our work focuses on the heterogeneity of the data model as well as the schema.

There have been efforts to rewrite queries against new models (such as, for a time, graphs) into queries against more traditional models (such as relations) [Jindal et al.(2014)]. Similarly, the translation of document queries (at the time in XPath or XQuery) to relational queries has received significant attention [Grust et al.(2003)]. However, this work focuses on mappings between one ideal and one imposed model and, therefore, has no need for Intermediate Representations.

Lastly, there is work on data-model integration using Intermediate Representations. The most recent approach is Obi-Wan [Buron et al.(2020)]. The authors suggest RDF/SPARQL as a Bridge Representation (though they do not use that term). However, their efforts lack generality: the approach is currently limited to querying data that straightforwardly maps between models (e.g. because it originated in the ideal model and has only been transformed into the imposed model). Our work aims to support arbitrary schema and queries.

7 CONCLUSION

The mismatch between ideal and imposed data model is faced by many applications. While this mismatch used to be a minor problem when only a few data models were in use, the ongoing proliferation of data models exacerbate the problem over time.

To address the problem, we propose a middle-ground approach between physical and virtual data integration, based on a "Bridge Representation" of queries that enables translation of queries between models when possible and transformation of data when necessary. We outline study requirements for such a Bridge Representation and outline a research agenda towards it. We demonstrate that, even when no perfect query translation is possible, queries can be overapproximated during translation, which reduces transformation cost compared to transforming all data.

REFERENCES

[sno([n. d.])] [n. d.]. Snowflake Vector Embeddings. https://docs.snowflake.com/en/user-guide/snowflake-cortex/vector-embeddings. Accessed: 2024-08-01.

[sql([n. d.])] [n. d.]. Vector Similarity Search with Azure SQL database and OpenAI. https://devblogs.microsoft.com/azure-sql/vector-similarity-search-with-azure-sql-database-and-openai/. Accessed: 2024-08-01.

[Buron et al.(2020)] Maxime Buron, François Goasdoué, Ioana Manolescu, and Marie-Laure Mugnier. 2020. Obi-Wan: ontology-based RDF integration of heterogeneous data. *PVLDB* (2020).

[Duggan et al.(2015)] Jennie Duggan, Aaron J Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. 2015. The bigdawg polystore system. *Sigmod Record* (2015).

[Funke et al.(2020)] Henning Funke, Jan Mühlig, and Jens Teubner. 2020. Efficient generation of machine code for query compilers. In *DaMoN*.

[Grust et al.(2003)] Torsten Grust, Maurice Van Keulen, and Jens Teubner. 2003. Staircase join: Teach a relational DBMS to watch its (axis) steps. In *VLDB*.

[Halevy(2001)] Alon Y Halevy. 2001. Answering queries using views: A survey. *VLDB Journal* (2001).

[Jindal et al.(2014)] Alekh Jindal, Praynaa Rawlani, Eugene Wu, Samuel Madden, Amol Deshpande, and Mike Stonebraker. 2014. Vertexica: your relational friend for graph analytics! (2014).

[Lattner et al.(2021)] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling compiler infrastructure for domain specific computation. In *IEEE CGO*.

[Lewis et al.(2020)] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NeurIPS* (2020).

[Li et al.(2023)] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281* (2023).

[Mackinnon et al.(1998)] Lachlan M Mackinnon, David H Marwick, and M Howard Williams. 1998. A model for query decomposition and answer construction in heterogeneous distributed database systems. *Journal of Intelligent Information Systems* (1998).

[Mohr-Daurat and Pirk(2021)] Hubert Mohr-Daurat and Holger Pirk. 2021. Homoiconicity For End-to-end Machine Learning with BOSS.. In *BICOD*.

[Mohr-Daurat et al.(2023a)] Hubert Mohr-Daurat, Xuan Sun, and Holger Pirk. 2023a. BOSS-An Architecture for Database Kernel Composition. *PVLDB* (2023).

[Mohr-Daurat et al.(2023b)] Hubert Mohr-Daurat, Giorgios Theodorakis, and Holger Pirk. 2023b. Hardware-Efficient Data Imputation through DBMS Extensibility. *PVLDB* (2023).

[Olteanu and Schleich(2016)] Dan Olteanu and Maximilian Schleich. 2016. Factorized databases. *SIGMOD Record* (2016).

[Palkar et al.(2017)] Shoumik Palkar, James J Thomas, Anil Shanbhag, Deepak Narayanan, Holger Pirk, Malte Schwarzkopf, Saman Amarasinghe, and Matei Zaharia. 2017. Weld: A Common Runtime for High Performance Data Analytics. *CIDR* (2017).

[Paradies et al.(2015)] Marcus Paradies, Wolfgang Lehner, and Christof Bornhövd. 2015. GRAPHITE: an extensible graph traversal framework for relational database management systems. In *SSDM*.

[Pedreira et al.(2022)] Pedro Pedreira, Orri Erling, Masha Basmanova, Kevin Wilfong, Laith Sakka, Krishna Pai, Wei He, and Biswapesh Chattopadhyay. 2022. Velox: meta's unified execution engine. *PVLDB* (2022).

[Pirk et al.(2016)] Holger Pirk, Oscar Moll, Matei Zaharia, and Sam Madden. 2016. Voodoo-a vector algebra for portable database performance on modern hardware. *PVLDB* (2016).

[Siek et al.(2001)] Jeremy G Siek, Lie-Quan Lee, and Andrew Lumsdaine. 2001. *The Boost Graph Library: User Guide and Reference Manual, The*. Pearson Education.

[Ulrich and Grust(2015)] Alexander Ulrich and Torsten Grust. 2015. The flatter, the better: Query compilation based on the flattening transformation. In *SIGMOD*.

[Wang et al.(2021a)] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021a. Milvus: A purpose-built vector data management system. In *SIGMOD*.

[Wang et al.(2021b)] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021b. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[William W. Cohen(2015)] Carnegie Mellon University William W. Cohen. 2015. Enron Email Dataset. https://www.cs.cmu.edu/~enron/ Accessed: 2024-11-07.

[Zhang and Chen(2018)] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *NeurIPS* (2018).