

# Optimizing Context-Enhanced Relational Joins

Viktor Sanca

EPFL

Lausanne, Switzerland

viktor.sanca@epfl.ch

Manos Chatzakis<sup>†</sup>

EPFL

Lausanne, Switzerland

emmanouil.chatzakis@epfl.ch

Anastasia Ailamaki\*

EPFL, Google

Switzerland, USA

anastasia.ailamaki@epfl.ch

**Abstract**—Collecting data, extracting value, and combining insights from relational and context-rich multi-modal sources in data processing pipelines presents a challenge for traditional relational DBMS. While relational operators allow declarative and optimizable query specification, they are limited to data transformations unsuitable for capturing or analyzing context. On the other hand, representation learning models can map context-rich data into embeddings, allowing machine-automated context processing but requiring imperative data transformation integration with the analytical query.

To bridge this dichotomy, we present a context-enhanced relational join and introduce an embedding operator composable with relational operators. This enables hybrid relational and context-rich vector data processing, with algebraic equivalences compatible with relational algebra and corresponding logical and physical optimizations. We investigate model-operator interaction with vector data processing and study the characteristics of the  $\mathcal{E}$ -join operator. Using an example of string embeddings, we demonstrate enabling hybrid context-enhanced processing on relational join operators with vector embeddings. The importance of holistic optimization, from logical to physical, is demonstrated in an order of magnitude execution time improvement.

**Index Terms**—analytics, vector embeddings, AI for database systems, query optimization, hardware-conscious processing

## I. INTRODUCTION

Relational databases allow declarative query specification and abstractions for logical and physical query plan optimizations. These optimizations include operator reordering via algebraic equivalences and heuristics and instantiating operators for resource-efficient and hardware-conscious execution on modern hardware. This allows ad-hoc query specification by abstracting out significant implementation details from the user and end-to-end optimization. As the main goal of relational analytical databases is to provide abstractions to large-scale processing and extraction of value from the data of interest, relational databases are designed for data types where a procedural way to process the data is possible to precisely specify, such as aggregating numerical values or processing strings with a well-specified pattern.

Still, many data sources are unsuitable for processing in a relational database and are typically only stored serialized in binary formats. These include documents, text, images, and other data sources of increasing value, driven by the advent of the Internet and mobile devices and services such as social media. Such data has a lot of human-understandable contexts:

<sup>†</sup> Author contributed during an internship at DIAS lab, EPFL.

\* Work done in its entirety at EPFL.

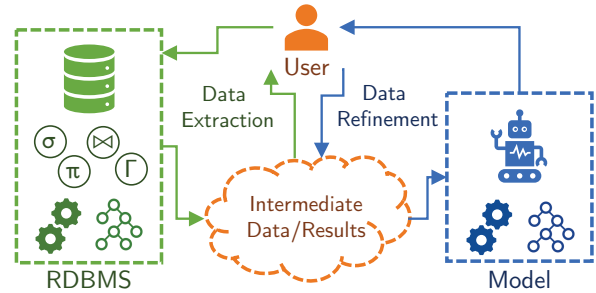


Fig. 1: Problem: Model-RDBMS data analysis requires user expertise, imperative tasks, and data movement specification.

the contents and number of objects in an image, the semantics of a string despite alternative spellings, typos, or tenses, all of which make this task impractical if not impossible to specify in traditional relational data analytics.

On the other hand, advancements in artificial intelligence and machine learning have allowed increasingly complex machine reasoning and performance in analyzing context-rich data such as images or text. Models such as BERT [1] and GPT [2] allow natural language processing, ResNet [3] object localization and detection, often available as Foundation Models [4] that are trained on web-scale data and further customizable and re-trainable for the particular task. To use those models, often based on Transformer architecture [5], analysts would instantiate the particular model, input data, and collect the output, using frameworks such as Tensorflow [6] or Pytorch [7], often in an isolated, task-specific setting. With the proliferation of embedding-based analytics, vector databases have recently gained traction, offering embedding storage and vector search, but with limited integration with traditional relational analytics or available operations over data.

While machine learning models can transform context-rich, multi-modal data into embeddings, coordinating the models and data processing pipelines is manual and imperative. Suppose an analyst wanted to analyze and extract insights from an RDBMS and use some data as input to the models as in Figure 1, which may be again input to an analytical query and models in a more complex analytical processing pipeline. Such a use case could combine the data from social media feeds with user reviews and transaction and analytics in an online retailer case, resulting in complex data processing pipelines, as the data of interest and value may not necessarily be tabular only, such as in canonical TPC-H, TPC-DS, or SSB [8] benchmarks.

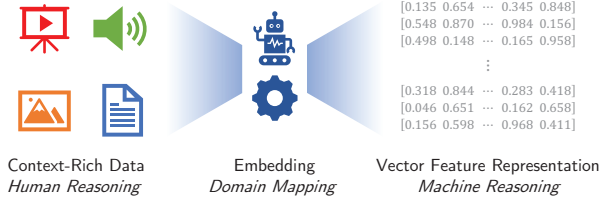


Fig. 2: Enabler: models embed context-rich data into common tensor representation, allowing automated processing.

With two independent components, RDBMS with relational data and Model with vector data, the user is back in the center of imperative program specification and data movement. This is not desirable, as a user must be an expert to fine-tune the queries, potentially perform data integration, correctly deploy and scale the queries to the hardware, orchestrate data movement, and specify the correct operator orders to prevent negatively impacting the performance. Decades of research and engineering in query optimization and execution engines allow hiding this complexity and is the key motivation to extend and make relational algebra the basis of emerging methods in multi-modal and context-rich processing.

Tightly integrated, expressive, and optimizable, hybrid vector-relational data management is part of our broader effort for the next generation of context-rich analytical engines [9]. The key enabler of this integration is that embedding models transform the domain of context-rich data into tensors as a common intermediate data representation, allowing strictly specified operations over high dimensional vectors such as similarity or analogies, providing a method to formalize the processing of data while preserving context in neural embedding space (Figure 2). A separation of concerns is established: model selection handles the multi-modality, data context, and semantics; the analytical engine optimizes and processes context-free data and tensors via exposed operators.

Thus, traditional relational operators have their counterparts and new physical and logical properties with neural-embedding-based vector processing. In this work, we investigate the case of context-enhanced join operation with model-operator interactions, which takes place over vector embeddings instead of only traditional relational data, and:

- Motivate and propose the capabilities of a context-enhanced join operator in Section II, and introduce and formalize the relational operator extension in Section III,
- Analyze the suitability of traditional join operator for the task of vector data processing, and propose a cost model, logical optimizations, and an alternative efficient tensor formulation for parallel execution of a join operator for processing neural embeddings in Section IV,
- Evaluate the physical and hardware optimizations we propose in Section V, and benchmark the operator implementation and characteristics in Section VI, showing the over orders of magnitude impact on the execution time and the importance of both logical and physical optimizations of vector-based join operations.

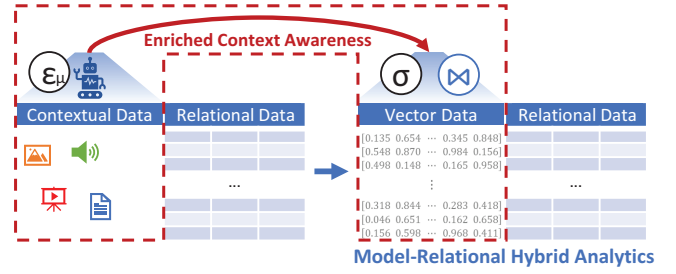


Fig. 3: Context-enhanced, model-relational analytics.

## II. MOTIVATION

There have been significant efforts to enable machine-automated understanding of context-rich data. The key idea behind neural embeddings is that the model ( $\mu$ ) learns how to transform the input data domain into high-dimensional vector space (Figure 2), where relationships between the data can be expressed using linear algebra expressions over vectors. Embedding models ( $\mathcal{E}_\mu$ ) take a human-understandable context-rich domain and map it into a machine-operable high dimensional vector (tensor) space.

### A. Extended Functionality: Joins Over Contextual Data

Model-driven embeddings transform data previously opaque to the relational data management system into context-free vectors, as illustrated in Figure 3. The separation of concerns between the model-driven context and uniform vector data representation enables defining expressions over vectors and tensors, such as semantic similarity using cosine distance, or other vector arithmetic operations, such as finding word analogies.

Broadly, this approach enables a fundamentally new way to join context-rich data such as strings, documents, or images by defining a corresponding model and vector expression. This formulation is closer to the traditional relational join, as a similarity predicate between two vectors can be formulated as an expression. We describe particular use cases for such context-extended relational data management systems.

#### 1) Semantic-Based Similarity Operations

Instead of having a human-in-the-loop or an expert system that performs dictionary-based or hard-coded rule-based similarity operations, neural embeddings allow the automation of similarity operations over many data types. A common tensor representation defines similarity joins as join condition expressions, such as cosine distance between the embedded vectors. The models fine-tune the functionality and context. After embedding the data and providing operators and expressions over tensors such as cosine distance, model-independent operations can be combined with the rest of the relational query plan.

#### 2) Online Data Cleaning

Strings or other context-rich data can be dirty or have rich semantics. If we consider words or sentences, they may have misspellings, alternative spellings, synonyms, or different tenses that all have the same meaning. Specifying all the rules

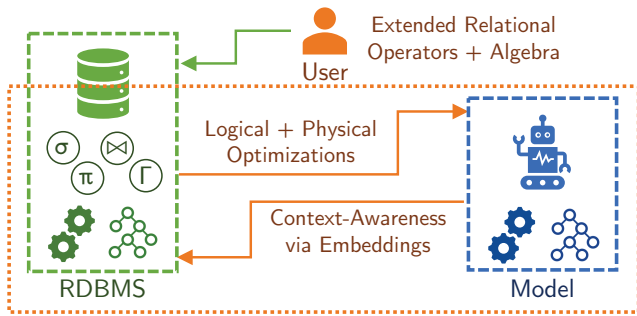


Fig. 4: Goal: Hybrid vector-relational operations are declarative transformation primitives amenable to query optimization.

to unify context is error-prone and difficult, while word embeddings can encompass such similarity using representation learning. Therefore, such operators can process such data on the fly without prior cleaning and only the data of interest, relying on embeddings and specified similarity thresholds for data integration and potentially performing post-verification steps.

### 3) Multi-Modal Data Processing

The data context is opaque to the execution engine, while the model selection and parameters give context and transform the data. By processing context-free tensors, relational engines provide a common optimization framework for multi-modal data driven by models, not relational engines. Join operations are not all-encompassing but provide a step towards unifying relational with model-driven data processing under a declarative and optimizable model.

## B. Integrating Vector Embeddings With Relational Operators

Data management systems support and simplify data processing with research and systems contributions and features such as transactions and concurrency control [10], [11], auto-tuning [12], hardware-conscious implementations [13]–[17], corresponding data structures [18], and query optimization with declarative interfaces to abstract out the system complexity from the end-user.

Instead of manual intermediate orchestration and system integration to combine and analyze multi-modal and context-rich data, involving different systems, data sources, and efficient operator reimplementations, we investigate how to extend traditional relational joins to support model-driven context with minimal system intrusions and build on top of existing judiciously modified abstractions. In particular, this means that the vectors are simply another data type over which expressions and operations can be defined. This makes index structures designed to store, maintain, and perform similarity search over tensors [19] compatible as physical access method options. Similarly, recent work has formulated traditional relational processing over tensors [20], where tensor processing platforms are used as analytical RDBMS to benefit from existing implementation while transforming the relational data and operations into tensor representation.

While there has been prior work to integrate model inference and learning with analytical engines [21], [22], our goal is complementary as we focus on how we can extend the relational model functionality with contextual data, as illustrated in Figure 4. Similarly, we expose co-optimization opportunities at logical, physical, and implementation levels and fine-grained system interactions [9].

## C. Holistic Optimization

Without loss of generality, suppose the data of interest are strings and dates stored in an RDBMS. Generally, one can consider other context-rich formats stored as binary objects with other relational data. To allow semantic similarity operations, such as matching strings that are synonyms, have misspellings, or different tenses, word embedding models transform strings into vectors, which are then comparable using cosine distance. While RDBMS could execute regex-like string expressions, mapping strings to embeddings allows capturing broader classes of similarity within a model. Note that the model can be trained and adapted for different datasets to adjust the notion of similarity, which the analyst selects.

We are interested in joining two tables over strings, where a condition over dates exists, making the queries selective on both tables. In a declarative setting, query specification requires embedding model information and the join condition expression, and the selectivity information from the relational column needs to propagate before the embeddings. Otherwise, the whole interaction may result in the user eagerly materializing all the data as in Figure 1, performing expensive embedding, and only then filtering. In more complex cases and ad-hoc queries, imperative specification and optimization are increasingly difficult.

Even with a declarative query with a logical plan with correct selectivities and operator orders, the word embedding model must interact with a join operator. Physical optimization must address this interaction and account for the tensor data format and the expressions suitable for comparing high-dimensional data. For example, while an equi-join over tensors could be implemented as a hash-join, more practical embedding comparisons, such as cosine distance, require algorithms such as nested-loop join for pair-wise comparisons and consider the join, operation, and model data structure access patterns in the algorithm and cost model.

Finally, from a hardware-conscious perspective, using many-dimensional vectors with relational operators designed and optimized for single-dimensional numerical data and judicious use of caches and memory hierarchy demands novel tradeoffs. A 100-dimensional tensor embedding will change the caching and execution patterns of traditional algorithms, and model embedding can incur computational or data access costs at a critical path of execution. Designing hardware-conscious algorithms represents a direction driven by novel model-database interactions. We aim to enable holistic optimization (Figure 4), starting from declarative specification through logical and physical optimization, and finally, hardware specialization to allow efficient execution.

**Takeaway** Neural embedding models transform the context-rich, multi-modal data into a common (per-model) tensor representation space. From the perspective of declarative relational processing, models provide separation between data semantics and context-less tensors as a model-parametrized projection operator. From there, relational operators perform operations such as cosine distance or vector transformations over tensors, amenable to query optimization via common abstractions and cost models that include model-operator interactions and physical optimizations aware of tensors and new computation and data access patterns co-designed for hardware. We analyze these behaviors and propose solutions that are aware of the new design space.

### III. CONTEXT-ENHANCED RELATIONAL JOIN OPERATOR

In this section, we start with formalizing the proposal of a relational operator extension to declaratively process context-rich data stored along traditional relational data, such as strings and text, that may be stored along with numerical or date attributes. We call this *hybrid model-relational processing*. This enhancement stems from the fact that the contextual data may need to be transformed and processed differently. However, compatibility with relational algebra and optimizations for processing purely relational data is required. Instead of using separate systems and manually orchestrating the data movement for processing using external programs or opaque UDFs, we propose a set of operations needed to express a join based on embedding the original data that is amenable to traditional query optimization.

#### A. Neural Embeddings

Neural embeddings and representation learning are rich and active research fields in machine learning. Images can be embedded with models such as ResNet [3], audio with PANNss [23], and text with Bert [1], word2Vec [24], Fast-Text [25], [26]. Foundation Models [4] offer an increasingly flexible way to specialize large models to a particular use case. It is important to mention those models can be tuned, as they learn representations from the training dataset through transfer learning [27] (e.g., starting from one of the foundation models) or re-training. In this work, we focus on and experiment with string embedding models. However, as embeddings are generally high-dimensional vectors, once in the embedding domain, the processing of this data is model- and input-data-type-agnostic, and the same principles and optimizations hold.

Processing embedded data allows automating semantic similarity using cosine similarity (or another distance) between the vectors. More complex relationships in the vector space are possible, such as analogies, such as country capitals (Switzerland  $\rightarrow$  Bern), (Greece  $\rightarrow$  **Athens**). The use of vectors necessitates interaction with linear algebra; therefore, the equations below outline definitions of cosine similarity over vectors and matrices. We will use them heavily in logical (Section IV) and physical (Section V) optimization phases.

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (\text{Cosine Similarity})$$

$$\frac{\mathbf{a}^{(1,d)} \cdot \mathbf{b}^{(1,d)}}{\|\mathbf{a}\| \|\mathbf{b}\|} = \frac{\sum_{i=1}^d a_i b_i}{\sqrt{\sum_{i=1}^d b_i^2} \sqrt{\sum_{i=1}^d b_i^2}} \quad (\text{Vector-Vector})$$

$$\frac{\mathbf{a}^{(1,d)} \cdot \mathbf{B}^{(m,d)}}{\|\mathbf{a}\| \|\mathbf{B}\|} = \left[ \frac{\mathbf{a} \cdot \mathbf{b}_i}{\|\mathbf{a}\| \|\mathbf{b}_i\|} \right]_{i=0}^m \quad (\text{Vector-Matrix})$$

$$\frac{\mathbf{A}^{(n,d)} \cdot \mathbf{B}^{(m,d)}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \left[ \left[ \frac{\mathbf{a}_i \cdot \mathbf{b}_j}{\|\mathbf{a}_i\| \|\mathbf{b}_j\|} \right]_{i=0}^n \right]_{j=0}^m \quad (\text{Matrix-Matrix})$$

#### B. Model-Operator Interaction

In our example, we focus on context awareness over strings so that common mistakes or semantically similar words are automatically captured. Rather than imposing user to strictly specify the rules for string similarity or clean the data ahead of time, we enable words such as (barbecue, barbecues, bbq, barbicue, grilling) that have similar semantics, to automatically be used with relational operator predicates without prior user intervention. The user should only specify the embedding model and a threshold distance parameter over cosine similarity calculation (Equation: Cosine Similarity). Instead of comparing two strings in their original domain, they are embedded. If the cosine similarity  $\cos(\theta)$  is larger than the specified threshold, the two strings are similar and should be matched. This avoids manual string processing and combining techniques, such as Locality Sensitive Hashing, individually limited to capturing only features such as misspellings.

A context-aware operator is supplemented with an embedding model ( $\mu$ ). In this case, when an operator receives strings, it embeds them and then performs the requested processing in the vector domain. Models can be selected based on the analyst’s needs, while often having desirable properties such as the capability of training and adapting to the desired similarity context. This interaction opens up design and optimization choices, such as how to mask or minimize the cost of embedding/model and overlap it with operator execution. We capture this interaction through relational algebra (Subsection III-C) and a cost model (Section IV) to allow holistic integration with the remainder of the query plan.

#### C. Relational Operators and Algebra

We introduce the embedding operator ( $\mathcal{E}$ ) using a model ( $\mu$ ), and relational algebra equivalences over selection ( $\sigma$ ) and  $\theta$ -join ( $\bowtie_\theta$ ) operations, compatible with traditional relational algebra definitions.

### 1) Selection

The selection operation applies predicate  $\theta$  over input tuples and returns only the tuples that satisfy the condition.

$$\sigma_{\theta}(R) = \{t \in R, \theta(t)\} \quad (\text{Selection with predicate } \theta)$$

To change the domain of input data, we allow mapping the input tuples (or a projection over the tuples for simple notation) using a model ( $\mu$ ) into vector space using embedding ( $\mathcal{E}$ ) operation.

$$\mathcal{E}_{\mu}(R) = \{t \in R, t \mapsto \mu(t)\} \quad (\text{Embedding with model } \mu)$$

For completeness and decoding of the embeddings and retrieving the context-rich data, an inverse operation  $\mathcal{E}^{-1}$  should also be defined, which is the standard component of encoder-decoder architectures, and semantically correct only for the same model  $\mu$ . Alternatively, a lookup table mechanism can maintain this object-embedding mapping.

$$\mathcal{E}_{\mu}^{-1}(\mathcal{E}_{\mu}(R)) = R \quad (\text{Decoding with model } \mu)$$

Combining embedding with selection allows the processing of tuples with a mixture of data formats. Some attributes may have traditional relational predicates, and some may require embedding and predicates using different metrics (such as cosine distance). Predicate pushdown and operation reordering can happen as soon as the attributes that predicates operate over are available.

$$\sigma_{\mathcal{E},\mu,\theta}(R) \Leftrightarrow \sigma_{\theta}(\mathcal{E}_{\mu}(R)) \Leftrightarrow \sigma_{\theta_{\mathcal{E}}}(\mathcal{E}_{\mu}(\sigma_{\theta_R}(R))) \quad (\mathcal{E}\text{-Selection})$$

### 2) Join

The join operation takes two relations and joins them over specified attributes using specified predicate conditions ( $\theta$ -join).

$$R \times S = \{(r, s), r \in R \wedge s \in S\} \quad (\text{Cartesian Product})$$

Joins are amenable to predicate pushdowns and reordering.

$$R \bowtie_{\theta} S \Leftrightarrow \sigma_{\theta}(R \times S) \quad (\theta\text{-Join Generalization})$$

We introduce embeddings to the generalized join definition and provide equivalences. Embeddings can be observed as a special projection operation that changes the domain.

$$R \bowtie_{\mathcal{E},\mu,\theta} S \Leftrightarrow \sigma_{\mathcal{E},\mu,\theta}(R \times S) \Leftrightarrow \quad (\mathcal{E}\text{-}\theta\text{-Join})$$

$$\sigma_{\theta}(\mathcal{E}_{\mu}(R) \times \mathcal{E}_{\mu}(S)) \Leftrightarrow \mathcal{E}_{\mu}(R) \bowtie_{\theta} \mathcal{E}_{\mu}(S)$$

Conversely, decoding the embeddings into their original domain is possible across the plan (Equation: Decoding with model  $\mu$ ).

**Takeaway** We formulate the context-enhanced operators by extending relational operators and algebra to allow declarative integration of embedding models with relational engines and optimizers. A hybrid setting enables declarative and systematic logical and physical optimizations, as depicted in the simple query in Figure 5, while providing semantic awareness using embeddings to separate concerns between models and engines.

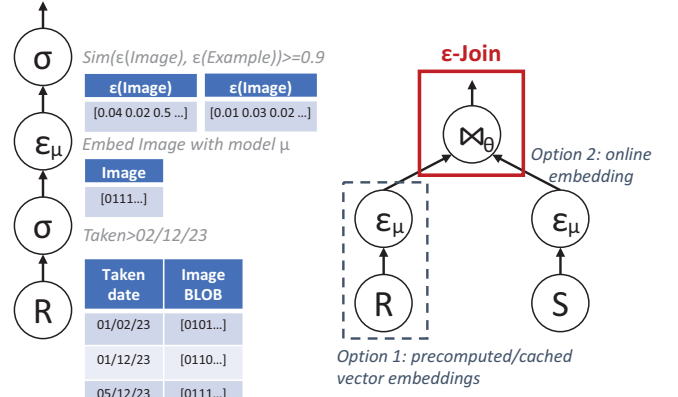


Fig. 5: Hybrid vector-relational query example, and the join operator which is the focus of the optimizations in this paper.

## IV. LOGICAL OPTIMIZATION

Starting from the extended algebra and operators, we present the logical optimization driven by model-operator interaction and tensors as a common intermediate data representation. In contrast to traditional optimizations and relational operator cost models, the two factors are different. First, since models may be on the critical path of execution, model embedding data access or computation time must be considered in addition to the relational operator's data access and processing cost. Second, embeddings change the data domain from traditionally *atomic* data types (prescribed by the 1<sup>st</sup> normal form) into dense high-dimensional vectors, which could benefit from optimizations in the domain of linear algebra. Still, regarding the 1<sup>st</sup> normal form, embeddings are not structured data but should be observed and processed atomically.

This changes the cost model and impacts the known characteristics of algorithms. For example, suppose an embedding is a 100-D vector. In that case, the cost of data movement and cache locality characteristics (spatial and temporal) are changed and must be evaluated in conjunction with the model's behavior and internal data structures.

### A. Cost Model

We outline the abstract cost model for the context-enhanced selection and join operations. For joins, we start by investigating the first available strategy: nested-loop join (NLJ). It is important to note that we focus on evaluating exact algorithms in our study. Since the distance we use is cosine similarity, hash-based approaches would yield approximate solutions similar to Locality Sensitive Hashing. Of course, if we were to use equi-joins, it would be possible to use traditional hash-joins, but there would be no benefit from using embeddings. Still, nested loop joins are a good fit as they can be formulated with good cache-locality, an important performance factor (Section VI) that does not incur random access over high dimensional data as every vector needs to be pair-wise compared using cosine distance.

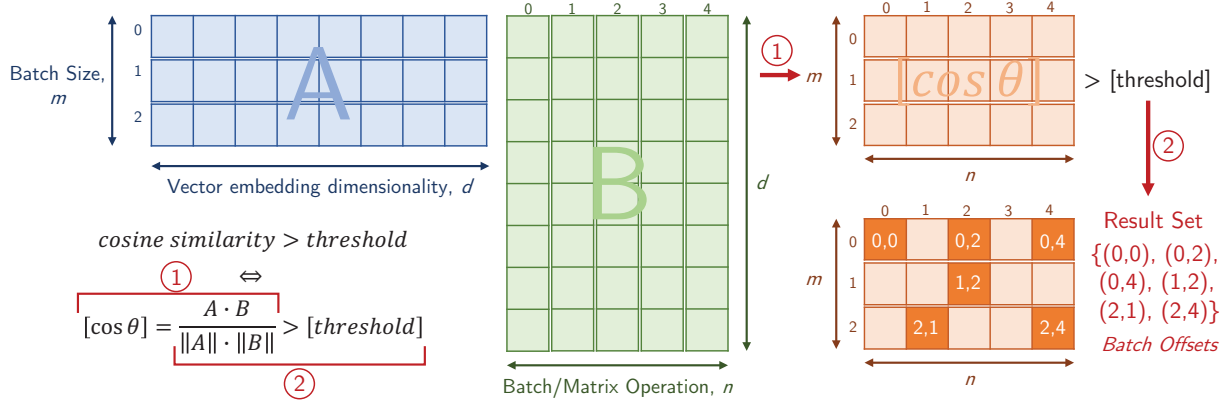


Fig. 6: Matrix formulation of  $\mathcal{E}$ -join allows scalable and cache-efficient execution over high-dimensional embeddings.

We outline the abstract cost model for selection and join below, where  $R$  and  $S$  are relations,  $|R|$  is the cardinality of relation  $R$ ,  $A$  represents the data access cost,  $M$  represents the model cost, and  $C$  is the computation cost.

Selection is an operation where input data is scanned, embedded, and the condition is applied over every input tuple, where each tuple incurs access, computation, and model cost:

$$\text{Cost}(\sigma_{\mathcal{E}, \mu, \theta}(R)) = |R| \cdot (A + M + C) \quad (\mathcal{E}\text{-Selection Cost})$$

By naively extending the Nested-Loop Join (NLJ) operation, it would scan both relations and perform pair-wise condition comparisons. In this implementation without considering model-relational interaction, model access would be performed per-processed tuple, which incurs quadratic model access cost. Considering that embedding models are computationally expensive, the following cost model shows the suboptimality:

$$\text{Cost}(R \bowtie_{\mathcal{E}, \mu, \theta} S) = |R| \cdot |S| \cdot (A + M + C) \quad (\mathcal{E}\text{-NL Join Cost})$$

Instead, by considering the characteristics of the nested-loop join, we observe that tuple embedding needs to happen only once per tuple from both relations. This can be performed as a precursor to the join operation or as a lazy embedding and data materialization strategy. By observing this behavior, the join results in a linear model cost with prefetching:

$$\text{Cost}(R \bowtie_{\mathcal{E}, \mu, \theta} S) = |R| \cdot |S| \cdot (A + C) + (|R| + |S|) \cdot M \quad (\mathcal{E}\text{-NLJ Prefetch Optimization})$$

This optimization is significant, as the model cost can span from random access to a lookup table (several times slower than sequential scan) to expensive computations over deep neural networks (data transfer and computation). From another perspective, if machine learning models are used as-a-service and paid for per embedding, this cost model conversely results in monetary savings compared to the potential suboptimal, manual implementation. Expressing embeddings as relational operator extensions allows logical optimization to occur in conjunction with other operators in the hybrid relational-embedding pipeline (selection pushdown, join reordering), such that the cardinality of the most costly part of the query plan will be reduced without explicit user intervention

or knowledge about specific interactions given a relational operator.

### B. Tensor Join Formulation

Since the computation of context-enhanced operators happens over dense high-dimensional embedding vectors, following the vector and matrix definitions of cosine distance in Subsection III-A, we present the tensor formulation of the dot product. It is important to highlight that cosine similarity is equivalent to the dot product with normalized input vectors.

The tensor formulation allows reasoning about the potential decomposition of the problem for parallel and cache-efficient execution beyond data parallelism, a basis for the physical optimization (Section V). This enables efficient and well-studied matrix-based algorithms for linear algebra in addition to the traditional relational algorithms. We present the block-matrix decomposition of the problem [28].

$$\mathbf{D}_{tv} = \sum_{i=1}^d \mathbf{R}_{ti} \mathbf{S}_{iv} \quad \begin{bmatrix} \mathbf{S}_{11} & \dots & \mathbf{S}_{1v} \\ \vdots & \ddots & \vdots \\ \mathbf{S}_{d1} & \dots & \mathbf{S}_{dv} \end{bmatrix} \rightarrow \mathbf{S}$$

$$\mathbf{R} \leftarrow \begin{bmatrix} \mathbf{R}_{11} & \dots & \mathbf{R}_{1d} \\ \vdots & \ddots & \vdots \\ \mathbf{R}_{t1} & \dots & \mathbf{R}_{td} \end{bmatrix} \quad \begin{bmatrix} \mathbf{D}_{11} & \dots & \mathbf{D}_{1v} \\ \vdots & \ddots & \vdots \\ \mathbf{D}_{t1} & \dots & \mathbf{D}_{tv} \end{bmatrix} \rightarrow \mathbf{D} = \mathbf{R}\mathbf{S}$$

(Block Matrix Dot Product Decomposition [28])

Given a  $(|R| \times \text{dim})$  matrix  $\mathbf{R}$  with  $t$  row partitions and  $d$  column partitions, and a  $(\text{dim} \times |S|)$  matrix  $\mathbf{S}$  with  $d$  row partitions and  $v$  column partitions that are compatible with the partitions of  $\mathbf{R}$ , dot product  $\mathbf{D} = \mathbf{R}\mathbf{S}$  can be formed block-wise, yielding  $\mathbf{D}$  as a  $(|R| \times |S|)$  matrix with  $t$  row partitions and  $v$  column partitions. We consider  $\mathbf{S}$  to be already transposed if the initial data layout is as of  $\mathbf{R}$ ; in other words, matrices  $\mathbf{R}$  and  $\mathbf{S}$  are compatible.

In particular, we partition the data along the tuple lines, not the dimensions, as illustrated in Figure 6 ①. Transforming the initial Nested-Loop Join into Tensor formulation enables applying linear algebra optimizations, in particular, matrix

multiplication algorithms to achieve better cache utilization of high-dimensional data, with well-understood parallelization using block-matrix decomposition.

This is compatible with and extends recent research on formulating relational operators for tensor processing run-times [29]. The cache is utilized better as, in contrast to NLJ, a matrix block (several vectors) can remain in the cache and be reused over many operations. Block-matrix partitioning allows for defining the processing granularity, which is significant when memory footprint needs to be constrained, allowing fine control of both the transfer and processing granularity of cosine-distance-based similarity operations, all while reducing redundant data transfers. In particular, this is a dense matrix operation which is highly computationally and data access optimized.

The next step is to map back to corresponding tuple pairs that satisfy the `threshold` condition, as in Figure 6 ②. It is sufficient to maintain the starting offsets of input relation partitions, so the result set constitutes a potentially sparse matrix of pairs that represent matrix batch offsets, driven by the predicate selectivity. This result can be considered as equivalent to using late materialization, and while sparse, it is more compact as tuples of offsets represent unique tensor identifiers. This is increasingly important when using novel memory hierarchies with fast but limited memory, such as high-bandwidth memory (HBM) [30].

**Takeaway** Formulating the cost model and alternative equivalent execution plans using linear algebra allows tuning the algorithms to the cost model and execution environment parameters, as high-dimensional vectors and model processing introduce data access, caching, and processing overheads. This is a mandatory step that enables further logical and physical optimizations, different from the ones suitable for traditional relational operators that process only single-dimensional data.

## V. PHYSICAL OPTIMIZATION

Modern data management systems are designed and optimized to efficiently utilize available hardware resources [13], [14], [17]. Equally, machine learning and linear algebra frameworks are designed with physical optimizations to allow fast and efficient execution over vector data [6], [7], [19], [31].

### A. Data-Parallel Execution

To benefit from many-core architectures, we outline the parallelization and hardware-conscious optimizations of the join algorithm. In contrast to the traditional Nested-Loop Join (NLJ) that allows exact cosine-distance-based joins, high-dimensional embedding vectors take up more space in the cache hierarchy. Consider a 32KB L1 cache, and we operate over 4-byte values. Using a 100-dimensional embedding vector, the L1 cache can store only 80 values, in contrast to 8000 for the single-dimensional data type. This necessitates cache-efficient implementation to benefit from the memory hierarchy. Furthermore, computing cosine distance over vectors requires more computation cycles than simply performing the regular

value-based operation. Thus, judicious use of hardware resources is necessary to speed up data access and computation.

#### 1) Data-parallelization strategies

Nested-Loop Join can be parallelized by partitioning the input relations and using a heuristic of keeping the smaller relation inside the inner loop to improve data and cache locality. This is a traditional and well-known NLJ algorithm implementation and optimization. In addition, we propose using the matrix (tensor) formulation (Figure 6) using linear algebra as an alternative to NLJ physical implementation. Matrix multiplication to obtain a dot product and normalize the vectors is embarrassingly parallel, in addition to having fine-grained control over partition size. In contrast to NLJ, matrix multiplication over dense vectors is a linear algebra operation with a better cache locality [32], [33], improving the use of the memory hierarchy in the presence of high-dimensional data, and using efficient matrix multiplication algorithms and linear-algebra frameworks.

#### 2) CPU Hardware Support

Traditionally, CPUs benefit from the main memory access locality. They are general-purpose compute units designed to process full-precision data types (e.g., 32-bit and 64-bit) that can support SIMD, such as with Intel AVX instructions. Recent AVX-512 [34] instruction set has introduced hardware support for half-precision data types, which allows processing up to 32 16-bit floating point numbers in a SIMD register. Furthermore, to support typical machine learning workloads, beyond providing hardware support for half-precision data types, CPUs such as 4th generation Intel Scalable Xeon Processors introduced specialized instruction sets (AMX) to accelerate matrix computations, along with limited capacity High-Bandwidth memory [30], [35]. In general, specialized instructions can accelerate the dense matrix computations. At the same time, low-latency access to memory enables optimizing the sparse matrix processing when processing the elements that satisfy the join predicate. As even the main memory is often limited or expensive, we will next discuss how to constrain the memory requirements of the tensor-based join formulation.

#### 3) SIMD Vectorization

Executing linear algebra operations such as cosine distance over dense vectors is compute-intensive and involves repeated operations over every vector embedding element. Since operations such as sum are repeated over every element of the logical vector, it is a natural fit for using single-instruction multiple-data instructions (SIMD). We use SIMD vectorization supported by hardware to speed up the arithmetic operations using fewer processing cycles using specialized registers and compute units, in conjunction with data-parallel partitioning for multi-core operator execution.

### B. Constraining the Memory Requirements

The tensor join formulation given in Figure 6 assumes a dot product operation between two matrices, a dense matrix linear algebra operation. This will result in a large intermediate state matrix of the same dimensions as the input relations.

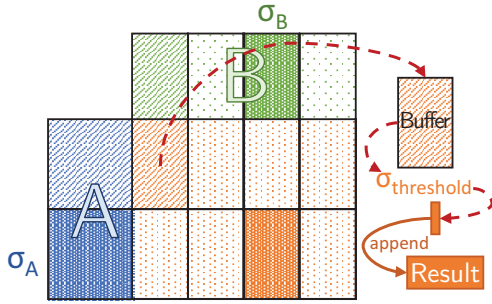


Fig. 7: Matrix partitioning constrains the memory requirement.

Despite joins being typically selective, which might reduce the matrix size, as in Figure 7, the intermediate state might still be too big to store and compute at once. Computing this matrix based on relations  $R$  and  $S$  would yield a  $|R| \times |S|$  memory requirement, which for 100k x 100k matrix that would store FP32 floating point values results in 40GB. While this matrix can be preserved to offset future computation, the primary purpose is to compute and return the offsets that satisfy the distance threshold requirement and return this result, which is typically a sparse operation. Thus, even for modest input relation sizes, this approach, in its current formulation, does not scale well concerning the memory requirements.

To resolve this issue, the previously presented matrix decomposition (Equation: Block Matrix Dot Product Decomposition [28]) allows scheduling the computation in batches and explicitly controlling the memory requirements based on the desired intermediate matrix size. This trades off memory for multiple invocations of the computation algorithm with smaller matrices, effectively computing the large one while pruning the intermediate sparse state after each matrix computation and comparison with the similarity threshold condition.

We illustrate this in Figure 7, where two relations  $A$  and  $B$  are joined over vectors. While the required memory requirement can be selectivity driven by other pushed-down relational predicates ( $\sigma_A$ ,  $\sigma_B$ ), this might not fit the available buffer budget. Thus, based on the available *Buffer* size, the input data can be partitioned arbitrarily by decomposing the input data along the vector tuple boundaries (not dimensions). The strict  $|A| \times |B|$  memory requirement becomes  $Buffer = |part(A)| \times |part(B)|$ , at the cost of several invocations of the algorithm that might reduce the overall performance by frequent data movement and lower cache locality.

**Takeaway** The physical operator design landscape encompasses implementation and hardware device characteristics-based decisions. Model-operator interactions only enrich and open a new design space. High-dimensional data contributes to reduced capacity of the memory hierarchy in comparison to common atomic data types found in relational data processing and requires rethinking cache-local implementations. With the increase in per-tuple compute cost, the strain is on both memory and compute resources, which invites the use of specialized hardware-conscious algorithms such as tensor join.

We start by demonstrating the functionality of using models as a driver of context-enhanced relational operations through the example of word embeddings. We then focus on the main performance evaluation of the proposed logical and physical optimizations, showing that a holistic approach is necessary to obtain a performing join algorithm.

**System** We implement our prototype operators and evaluation in a standalone system in C++ and use Intel AVX instructions for SIMD execution. Tensor formulation benchmarks use Intel oneAPI Math Kernel Library for CPU-aware and efficient BLAS-based linear algebra operations.

**Hardware Setup** We run the end-to-end and scalability experiments on a two-socket Intel Xeon Gold 5118 CPU (2 x 12-core, 48 threads) and 384GB of RAM. All experiments are with in-memory data; experiments with synthetic data use the same random number generator seed for reproducibility.

#### A. Enhancing Operator Context via Word Embeddings

In our study, we use the example of word embeddings that transform input strings into high-dimensional vectors. We show the context-awareness functionality that word embeddings allow and note that embedding models can be fine-tuned and replaced to support different notions of similarity. Conversely, there are embedding models to support different data modalities. Still, the intermediate data representation of an embedding is a context-free vector that operators process independently of the particular model, on top of which we base our analysis. The proposed optimizations of our approach are independent by design and principled in approach due to the separation of concerns between the model, which produces vectors, and the operator performing the join over context-free vectors.

**Embedding Model** We use FastText [25], [26] as a model ( $\mu$ ) for string embeddings, which has the desirable properties that it can be trained and adapted to the context, it supports out of vocabulary word embedding and is resilient to misspellings. A context-aware operator is supplemented with an embedding model. In this case, when an operator receives strings, it embeds them using FastText and then performs the requested processing in the vector domain.

**Dataset** We train a 100-dimension embedding model over a subset of Wikipedia dataset [36], cleaned of stopwords, using a subset of 1M strings from the dataset to test the similarity using the model. We show the nearest vectors to sample words as the strings are embedded into a high-dimensional vector space. We then decode the vectors back into string and present sample semantic matches in Table I. The model has learned semantics and context from the Wikipedia dataset. To fine-tune the model, it is possible to specialize the embedding models with other domain-specific datasets.

It is important to mention that models allow automated semantic matching, and the strings are not materialized or retrieved during operations in an intermediate step. The computation entirely happens on embedded data, and only positive matches are retrieved. It is possible to decode the embeddings,



TABLE I: Semantic Matching using FastText trained on Wikipedia dataset, 100-D embeddings, sample words.

Word	Top-15 Model Matches
dbms	rdbms, nosql, dbmss, postgresql, rdbmss, sql, dbmses, sqllite, dataflow, ordbms, oodbms, couchdb, mysql, ldap, oltp
postgres	postgre, postgresql, openvt, dbms, rdbmss, sqllite, dbmss, odbc, backend, rdbms, rdbmses, postgis, openvp, couchdb, mysql
animal	animals, felines, human, bovines, equines, dogs, nonhuman, ferrets, rabbits, chickens, anthropods, bovine, anthropod, mammal, equine
dog	dogs, poodle, doberman, sheepdog, puppies, dachshund, hound, bullmastiff, retriever, pinschers, dobermans, puppy, bullmastiffs, chickenhound, dachshunds
clothes	dresses, clothing, garments, underwear, bedclothes, undergarments, towels, underweares, scarves, shoes, nightgowns, clothings, bathrobes, underclothes

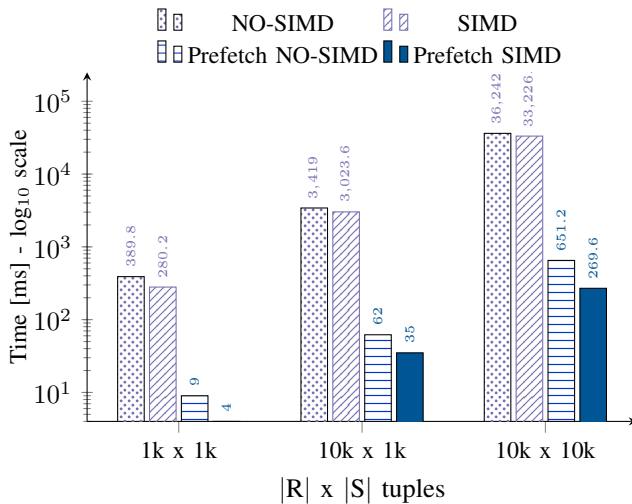


Fig. 8: The impact of logical and physical optimization on NLJ formulation. 100-D vectors, 48 threads.

for example, based on their offset in the input relation and processing the embedding using standard encoder-decoder model architecture.

This model aimed to detect synonyms, semantic and related matches, and plural forms of the words without external user specification. The only parameter in the case of a join with cosine distance would be a single threshold parameter. This allows relational operators normally operating over sample strings (i.e., Word column in Table I) to perform matches with strings on the right in the embedding domain without humans in the loop or experts that would create and specify strict rules. Such models can work by providing positive match examples that could infer the correct cosine distance threshold parameter.

### B. NLJ Formulation: Logical Optimization

As introduced in Section IV, we extend the traditional relational join formulation by embedding vector processing and retrieval. We evaluate the impact of logical optimization of vector prefetching and physical optimization using SIMD in Figure 8. This experiment validates the cost difference between the naive join extension (Equation:  $\mathcal{E}$ -NL Join Cost)

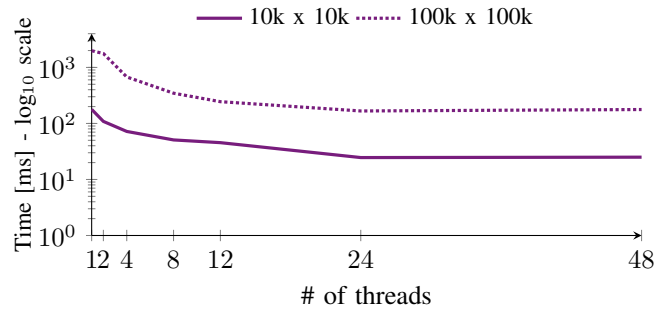


Fig. 9: Optimized NLJ formulation scalability, 100-D vectors.

and the one aware of the vector retrieval (Equation:  $\mathcal{E}$ -NLJ Prefetch Optimization). Not prefetching the embeddings incurs quadratic model access costs validating the cost model, resulting in orders of magnitude slower execution time.

This emphasizes the importance of analyzing, exposing, and optimizing model-operator interactions. Despite using the same hardware resources, including separate experiments with and without SIMD, the main bottleneck is not computational but access-pattern-related and algorithmic. With the wrong holistic operator formulation, faster hardware cannot correct the suboptimal formulation, as may happen with imperative operator specification by a non-expert user. In this case, the optimal strategy of prefetching the embeddings first and then joining, despite having two separate tasks, allows faster execution time. SIMD instructions improve the execution time 2x, indicating a computational bottleneck that additional hardware instructions reduce, while this is not possible in the non-prefetch, sub-optimal formulation.

In Figure 9, we show the scalability of the proposed logical optimization with prefetching. We compare model prefetching over two input relations sized 10k x 10k, and 100k x 100k, which results in  $10^8$  and  $10^{10}$  computations, respectively. Notice the *log*-scale of the figure. Using the improved NLJ cost model formulation and comparing the execution time between the two input sizes, execution time scales linearly instead of quadratically as in the non-optimized case and by an expected factor of  $10^2$ .

**Takeaway.** Logical operator optimizations and task orchestration are crucial to removing algorithmic bottlenecks. Allocating more resources cannot scale and is wasteful before resolving an algorithm’s logical costs and overheads.

### C. NLJ Formulation: Physical Optimization

We focus next on the physical optimizations and demonstrate the scalability of CPU execution and physical and logical optimizations of NLJ formulation presented in Section V. First, we investigate the impact of SIMD vectorization (Figure 10). We enable hyperthreading (24 physical, 48 logical cores), affinitize threads to cores (2 threads will run on 1 physical core, 4 on 2, etc.), and run the NLJ formulation of 10k x 10k relation size input with 100-dimensional embeddings. The processor has AVX-512 registers that can simultaneously fit 16 32-bit floating-point values simultaneously. The average improvement is 5.36x, indicating non-computational overheads

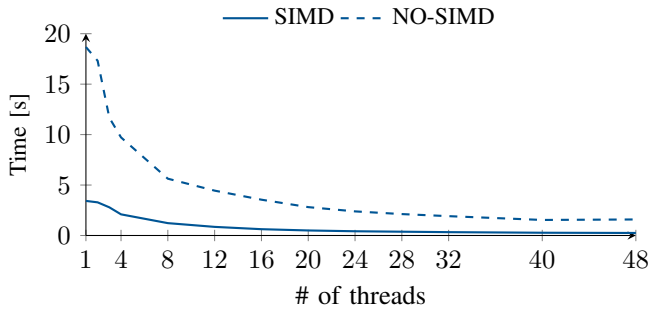


Fig. 10: NLJ formulation scalability and impact of SIMD, 10k x 10k join input relations, 100-D vectors.

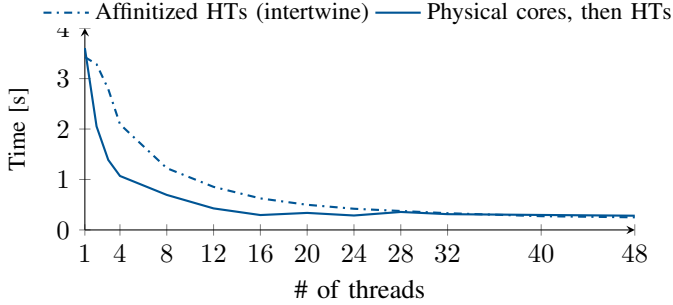


Fig. 11: NLJ formulation scalability with SIMD, prioritizing physical cores, 10k x 10k input relations, 100-D vectors.

during vectorization but improved execution time using available hardware intrinsics.

We use two strategies to investigate the effect of affinitizing threads to cores (Figure 11). First, we assign the physical cores to the thread pool, followed by hyper-threads. Second, we affinitize threads and hyper-threads to cores (2 threads will run on 1 physical core, 4 on 2, until 24 cores/48 threads). While affinitized strategy scales with added cores, physical cores scale faster, and hyper-threads do not contribute to improved execution time. This corroborates that dense vector linear algebra is computationally heavy, and physical cores cannot simultaneously benefit from scheduling a hyper-thread. At the shift from physical to added logical cores, we see an increase in the physical, then HTs strategy due to data-parallel execution at 28 threads, as 28 partitions were assigned to 24 physical cores, creating 4 stragglers. Finally, using all the available resources (after 24 threads), the result is the same.

Finally, we evaluate the impact of different input relation sizes (in tuples) over 100-D, 32-bit embeddings over 48 threads and investigate the impact of physical and logical optimizations using the NLJ formulation. In this experiment (Figure 12), we investigate the effects of input sizes, number of computations, and ordering of input relations of context-enhanced NLJ implementation. First, the execution time scales linearly with the number of computations/operations performed, according to the cost model (Equation:  $\mathcal{E}$ -NLJ Prefetch Optimization). Second, we validate that to achieve improved execution time due to cache locality, smaller relation should still be the inner loop, as in the traditional nested-loop-join, while using 100-D vectors for cosine distance

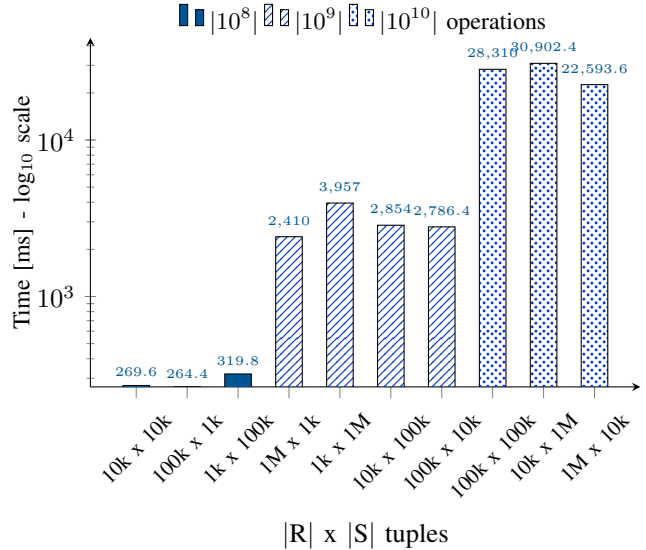


Fig. 12: Optimized NLJ formulation with varying input relation sizes, 100-D vectors, 48 threads.

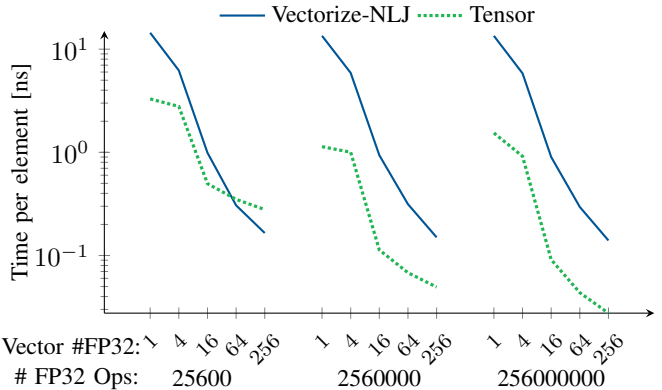


Fig. 13: Physical optimization. The tensor strategy (green) pays off in larger inputs compared to NLJ (blue).

computation. Despite more expensive per-vector computations, data access patterns still play an important role, impacting our experiment’s performance by up to  $\sim 35\%$  (at  $10^{10}$  operations).

**Takeaway.** Logical and physical optimizations of the NLJ formulation with vectors enable reducing the overheads by orders of magnitude from the initial vector join extension. Still, the approaches we proposed until now optimize for vector execution without explicitly considering the high dimensionality and similarity operations over individual tuples. The tensor formulation, which we will evaluate next, addresses this issue.

#### D. Tensor Formulation: The Holistic Vector-Join Optimization

Instead of applying optimized computation on individual vector operations in the NLJ, in Figure 6, we propose batching multiple vector tuples together in a tensor join formulation using optimized matrix computation. The key enabler and difference is that BLAS matrix operations are highly optimized for the cache locality that the simple NLJ imposed in its formulation. We evaluate this physical optimization proposed in

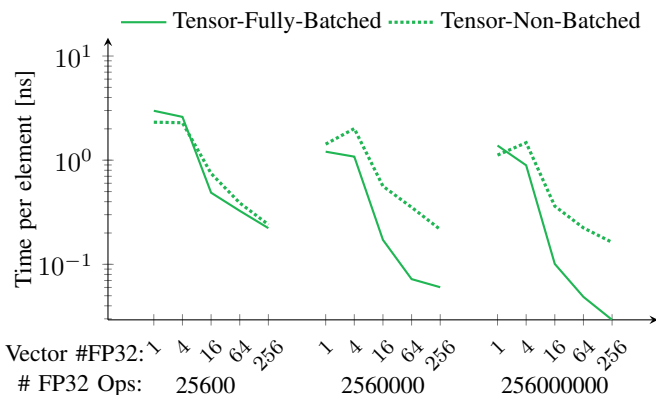


Fig. 14: The impact of vector batching. Non-batched indicates that one of the join inputs is processed one vector at a time.

Section V, evaluating whether the tensor formulation improves the per-vector-element processing time. We compare two strategies, running the fully optimized NLJ against the Tensor formulation. For this, we vary two factors: the total number of floating point numbers processed (#FP32 Ops) and how many floating point numbers represent an individual vector (vector dimensionality, Vector #FP32). Figure 13 summarizes the findings, where three data clusters are based on the number of operations, refined by individual vector size. In other words, for the 25600 case with dimensionality 1, there are  $25600/1$  tuples joined, equally balanced in two relations, indicating  $\sqrt{25600/1} = 160$  tuples per input relation. Similarly, to obtain the number of tuples for the case of dimensionality 256,  $\sqrt{25600/256} = 10$ . We use the per-FP32 breakdown as a unifying metric across the input size and dimensionality. First, we notice the benefit of vectorization with increased vector size, where specialized hardware operations improve the per-tuple performance. Second, pushing this boundary beyond per-tuple-vector but to a whole tuple-vector-batch (Tensor), when sufficient computation can benefit from the cache locality, this approach significantly improves the execution time. In particular, the Tensor approach was slower only in case there were  $\sqrt{25600/64} = 20$  and  $\sqrt{25600/256} = 10$  tuples to join.

Batching the vectors together in the tensor formulation is the key to reducing unnecessary data movement. We demonstrate the impact of batching in Figure 14, where the BLAS-matrix operations are used with one fully batched relation. At the same time, the other is loaded vector-by-vector, repeated as many times as there are tuples. An alternative is where both relations are fully batched. While inefficiencies are not noticeable with very small input sizes, batching becomes increasingly significant for scalability as the input grows.

Still, as explained in Subsection V-B, batching too many vectors together in large tensors simultaneously comes at a prohibitive memory cost. We propose using mini-batches partitioned across tuple boundaries (Figure 7) that can still benefit from the improved linear algebra algorithms and data locality. The impact of batching is presented in Figure 15. We run the tensor join formulation over 100k x 100k, 100-D input using 48 threads. The No Batch case runs the join on the

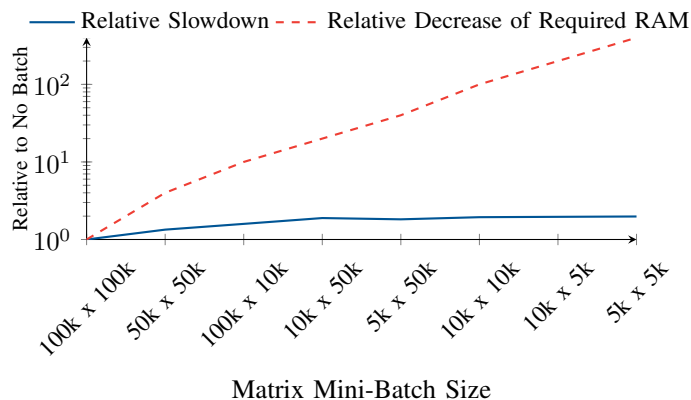


Fig. 15: Batch size impact on memory requirements and execution time. 100k x 100k, 100-D input (No Batch case).

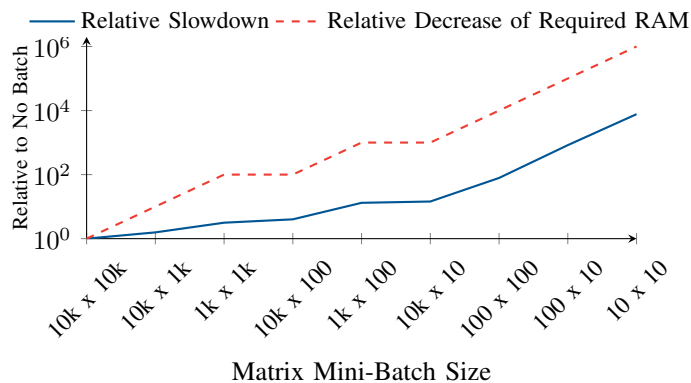


Fig. 16: Fine-grained batch size impact on memory requirements and execution time. 10k x 10k, 100-D (No Batch case).

whole input at once. At the same time, the experiment focuses on memory footprint reduction and the computational price to pay when various mini-batches are used. While there is a negligible relative slowdown due to some added data movement and repeated operations, there is a significant benefit due to the reduction of the necessary memory. However, taking mini-batching to extremely small cases results in an NLJ formulation performance where the computation boundary is an individual vector. In such cases, the slowdown scales along with the reduced memory requirement, as shown in Figure 16, as the available computational resources (SIMD, cores) remain idle, and explicit data movement becomes predominant.

Finally, we compare the NLJ with the Tensor formulation end-to-end execution time in Figure 17. While the execution time of both algorithms scales approximately linearly when increasing the input relation size, the algorithm optimizations enabled by batching vectors into tensors opened linear algebra-based optimizations. This enables further execution time gains at almost an order of magnitude improvement across various input sizes.

**Takeaway.** Holistic optimization of the join algorithm with vector inputs is necessary to enable fast and efficient computation. This led to removing model-operator overheads, tuning the individual and batched vector computation and access patterns aware of the underlying hardware capabilities.

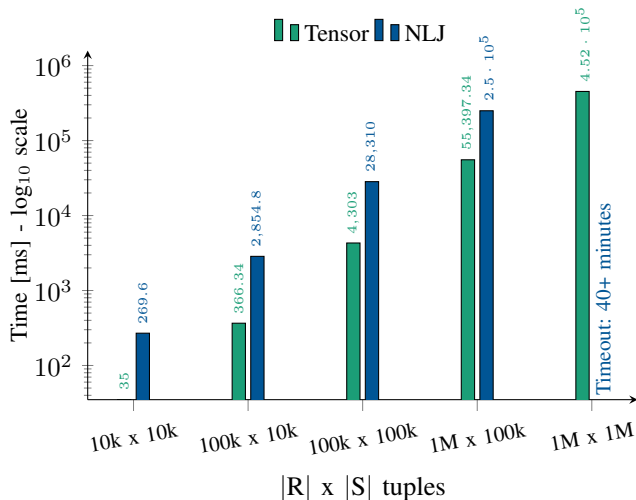


Fig. 17: Tensor join vs. NLJ formulation, 100-D, 48 threads.

## VII. RELATED WORK

This section outlines the related work and compares and places our approach in the rich design space of prior research.

### A. Machine Learning for Databases

Machine learning for databases has been a topic of research where structural components of DBMS get enhanced using findings from the ML community. Learned indexes [37] avoid data structure traversal by learning the data distribution information and optimizing data access. From a systems perspective, using tensor processing frameworks for traditional relational processing has also recently been proposed [20].

Our approach is similar as we propose using ML embedding models to provide context to data traditionally opaque to relational DBMS. We propose a general framework for extending and analyzing relational operators using novel model-database interactions.

### B. Databases for Machine Learning

On the other hand, databases for machine learning focus on applying or integrating machine learning components with systems. Frameworks such as Tensorflow [6] or Pytorch [7] are efficient, hardware-conscious dataflow engines. There has been recent work on developing vector-specialized databases [19] typically based on indexes for efficient high-dimensional similarity search [31]. Still, such engines often lack a DBMS’s expressiveness, functionality, and analytical operations for more complex data analysis, which would force users to write imperative code to integrate different siloed system components.

### C. String Similarity Joins

We have proposed a join operator that functionally resembles a string similarity join. Traditional string similarity techniques require exact similarity specification using edit distance or q-grams as token-based string similarity [38]. Similarly, locality-sensitive hashing techniques [39] exist for

approximate joins. These approaches are adequate for finding misspellings and limited token-based differences.

In contrast, we propose using word embedding models capable of identifying misspellings, different tenses, and semantic similarity based on training and fine-tuning training dataset and parameters [25], [26]. Through the separation of concerns, from the perspective of DBMS, string similarity join has a tensor-based input with cosine distance and threshold as parameters. At the same time, the embedding model handles the string semantics and context and transforms the input into context-free embeddings for RDBMS to process. Furthermore, our approach extends the notion of similarity to context-rich data for which embedding models trained for correct similarity semantics exist.

### D. Representation Learning

The significant body of work in representation learning is the key enabler of context-rich relational operators. It allows for transforming the human-centric, context-rich data representations into machine-centric formats amenable to automated processing. We combine ML-based embedding models with relational operators and analyze end-to-end interactions, from logical to physical optimizations. Such models allow masking and transforming contextual data into embeddings as ubiquitous data representations that can be processed by extending RDBMS with tensor-based operators.

A rich research area in machine learning drives embedding models that support other context-rich data formats beyond strings, equally transforming the input into context-free embeddings. Enabling multi-modality through model-operator interactions in a topic of future research, where models such as ResNet [3] can be used for images or PANNs [23] for audio processing. Models trained on web-scale data exist as Foundation Models [4], that can be re-trained and adapted for a specific task and dataset.

## VIII. CONCLUSION AND FUTURE DIRECTIONS

Data management systems support analysts with modern data processing tools. With strong results in automating context-rich processing from the machine learning community, such individual components would require imperative integration in complex analytical data pipelines. We propose instead a context-rich join operation that integrates embedding processing with relational operators based on the key observation of the separation of concerns. Embedding models are designed to handle context, while RDBMS needs to provide a declarative context-free interface based on extended relational algebra that allows logical and physical optimizations of the interactions between operators and models. With the common tensor data representation, we analyze the behavior of a join operator and propose relational algebra extensions, logical and physical optimizations, and introduce optimizations for efficient execution. We evaluate our cost model and show the impact holistic optimizations have on the execution time, resulting in orders of magnitude differences across logical, physical, and hardware optimizations.

## REFERENCES

- [1] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: <https://doi.org/10.18653/v1/n19-1423>
- [2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. [Online]. Available: <https://doi.org/10.1109/CVPR.2016.90>
- [4] R. Bommasani, D. A. Hudson, E. Adeli, R. B. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. S. Chatterji, A. S. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. D. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. S. Krass, R. Krishna, R. Kudithipudi, and et al., "On the opportunities and risks of foundation models," *CoRR*, vol. abs/2108.07258, 2021. [Online]. Available: <https://arxiv.org/abs/2108.07258>
- [5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
- [6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., "Tensorflow: a system for large-scale machine learning," in *Osdi*, vol. 16, no. 2016. Savannah, GA, USA, 2016, pp. 265–283.
- [7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.
- [8] P. E. O’Neil, E. J. O’Neil, and X. Chen, "The star schema benchmark (ssb)," *Pat*, vol. 200, no. 0, p. 50, 2007.
- [9] V. Sanca and A. Ailamaki, "Analytical engines with context-rich processing: Towards efficient next-generation analytics," in *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2023, pp. 3699–3707. [Online]. Available: <https://doi.org/10.1109/ICDE55515.2023.00298>
- [10] L. Zhang, M. Butrovich, T. Li, A. Pavlo, Y. Nannapaneni, J. Rollinson, H. Zhang, A. Balakumar, D. Biales, Z. Dong, E. J. Eppinger, J. E. Gonzalez, W. S. Lim, J. Liu, L. Ma, P. Menon, S. Mukherjee, T. Nayak, A. Ngom, D. Niu, D. Patra, P. Raj, S. Wang, W. Wang, Y. Yu, and W. Zhang, "Everything is a transaction: Unifying logical concurrency control and physical data structure maintenance in database management systems," in *CIDR 2021, Conference on Innovative Data Systems Research*, 2021. [Online]. Available: [https://db.cs.cmu.edu/papers/2021/cidr2021\\_paper06.pdf](https://db.cs.cmu.edu/papers/2021/cidr2021_paper06.pdf)
- [11] A. Kemper and T. Neumann, "Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots," in *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, S. Abiteboul, K. Böhm, C. Koch, and K. Tan, Eds. IEEE Computer Society, 2011, pp. 195–206. [Online]. Available: <https://doi.org/10.1109/ICDE.2011.5767867>
- [12] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang, "Self-driving database management systems," in *CIDR 2017, Conference on Innovative Data Systems Research*, 2017. [Online]. Available: <https://db.cs.cmu.edu/papers/2017/p42-pavlo-cidr17.pdf>
- [13] T. Neumann, "Efficiently compiling efficient query plans for modern hardware," *Proc. VLDB Endow.*, vol. 4, no. 9, pp. 539–550, 2011. [Online]. Available: <http://www.vldb.org/pvldb/vol4/p539-neumann.pdf>
- [14] P. Chrysogelos, M. Karpathiotakis, R. Appuswamy, and A. Ailamaki, "Hetexchange: Encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines," *Proc. VLDB Endow.*, vol. 12, no. 5, pp. 544–556, 2019. [Online]. Available: <http://www.vldb.org/pvldb/vol12/p544-chrysogelos.pdf>
- [15] T. Neumann and M. J. Freitag, "Umbra: A disk-based system with in-memory performance," in *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org, 2020. [Online]. Available: <http://cidrdb.org/cidr2020/papers/p29-neumann-cidr20.pdf>
- [16] T. Kersten, V. Leis, A. Kemper, T. Neumann, A. Pavlo, and P. A. Boncz, "Everything you always wanted to know about compiled and vectorized queries but were afraid to ask," *Proc. VLDB Endow.*, vol. 11, no. 13, pp. 2209–2222, 2018. [Online]. Available: <http://www.vldb.org/pvldb/vol11/p2209-kersten.pdf>
- [17] M. Zukowski, M. van de Wiel, and P. A. Boncz, "Vectorwise: A vectorized analytical DBMS," in *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, A. Kementsietsidis and M. A. V. Salles, Eds. IEEE Computer Society, 2012, pp. 1349–1350. [Online]. Available: <https://doi.org/10.1109/ICDE.2012.148>
- [18] S. Idreos, K. Zoumpatianos, B. Hentschel, M. S. Kester, and D. Guo, "The data calculator: Data structure design and cost synthesis from first principles and learned cost models," in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, G. Das, C. M. Jermaine, and P. A. Bernstein, Eds. ACM, 2018, pp. 535–550. [Online]. Available: <https://doi.org/10.1145/3183713.3199671>
- [19] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021. [Online]. Available: <https://doi.org/10.1109/TBDDATA.2019.2921572>
- [20] A. Gandhi, Y. Asada, V. Fu, A. Gemawat, L. Zhang, R. Sen, C. Curino, J. Camacho-Rodríguez, and M. Interlandi, "The tensor data platform: Towards an ai-centric database system," 2023. [Online]. Available: <https://www.cidrdb.org/cidr2023/papers/p68-gandhi.pdf>
- [21] Q. Lin, S. Wu, J. Zhao, J. Dai, F. Li, and G. Chen, "A comparative study of in-database inference approaches," in *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2022, pp. 1794–1807. [Online]. Available: <https://doi.org/10.1109/ICDE53745.2022.00180>
- [22] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar, "The madlib analytics library or MAD skills, the SQL," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1700–1711, 2012. [Online]. Available: [http://vldb.org/pvldb/vol5/p1700-joehellerstein\\_vldb2012.pdf](http://vldb.org/pvldb/vol5/p1700-joehellerstein_vldb2012.pdf)
- [23] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, "Panns: Large-scale pretrained audio neural networks for audio pattern recognition," *IEEE ACM Trans. Audio Speech Lang. Process.*, vol. 28, pp. 2880–2894, 2020. [Online]. Available: <https://doi.org/10.1109/TASLP.2020.3030497>
- [24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013. [Online]. Available: <http://arxiv.org/abs/1301.3781>
- [25] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, 2017. [Online]. Available: [https://doi.org/10.1162/tacl\\_a\\_00051](https://doi.org/10.1162/tacl_a_00051)
- [26] B. Edizel, A. Piktus, P. Bojanowski, R. Ferreira, E. Grave, and F. Silvestri, "Misspelling oblivious word embeddings," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language*

- Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 3226–3234. [Online]. Available: <https://doi.org/10.18653/v1/n19-1326>
- [27] Y. Qi, D. S. Sachan, M. Felix, S. Padmanabhan, and G. Neubig, “When and why are pre-trained word embeddings useful for neural machine translation?” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, M. A. Walker, H. Ji, and A. Stent, Eds. Association for Computational Linguistics, 2018, pp. 529–535. [Online]. Available: <https://doi.org/10.18653/v1/n18-2084>
- [28] K. B. Petersen and M. S. Pedersen, “The matrix cookbook,” nov 2012, version 20121115. [Online]. Available: <http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html>
- [29] D. He, S. C. Nakandala, D. Banda, R. Sen, K. Saur, K. Park, C. Curino, J. Camacho-Rodríguez, K. Karanasos, and M. Interlandi, “Query processing on tensor computation runtimes,” *Proc. VLDB Endow.*, vol. 15, no. 11, pp. 2811–2825, 2022. [Online]. Available: <https://www.vldb.org/pvldb/vol15/p2811-he.pdf>
- [30] V. Sanca and A. Ailamaki, “Post-moore’s law fusion: High-bandwidth memory, accelerators, and native half-precision processing for cpu-local analytics,” in *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023*, ser. CEUR Workshop Proceedings, R. Bordawekar, C. Cappiello, V. Efthymiou, L. Ehrlinger, V. Gadepally, S. Galhotra, S. Geisler, S. Groppe, L. Gruenwald, A. Y. Halevy, H. Harmouch, O. Hassanzadeh, I. F. Ilyas, E. Jiménez-Ruiz, S. Krishnan, T. Lahiri, G. Li, J. Lu, W. Mauerer, U. F. Minhas, F. Naumann, M. T. Özsu, E. K. Rezig, K. Srinivas, M. Stonebraker, S. R. Valluri, M. Vidal, H. Wang, J. Wang, Y. Wu, X. Xue, M. Zait, and K. Zeng, Eds., vol. 3462. CEUR-WS.org, 2023. [Online]. Available: <https://ceur-ws.org/Vol-3462/ADMS1.pdf>
- [31] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, K. Yu, Y. Yuan, Y. Zou, J. Long, Y. Cai, Z. Li, Z. Zhang, Y. Mo, J. Gu, R. Jiang, Y. Wei, and C. Xie, “Milvus: A purpose-built vector data management system,” in *SIGMOD ’21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, G. Li, Z. Li, S. Idreos, and D. Srivastava, Eds. ACM, 2021, pp. 2614–2627. [Online]. Available: <https://doi.org/10.1145/3448016.3457550>
- [32] K. Goto and R. A. van de Geijn, “Anatomy of high-performance matrix multiplication,” *ACM Trans. Math. Softw.*, vol. 34, no. 3, pp. 12:1–12:25, 2008. [Online]. Available: <https://doi.org/10.1145/1356052.1356053>
- [33] T. M. Smith, R. A. van de Geijn, M. Smelyanskiy, J. R. Hammond, and F. G. V. Zee, “Anatomy of high-performance many-threaded matrix multiplication,” in *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014*. IEEE Computer Society, 2014, pp. 1049–1059. [Online]. Available: <https://doi.org/10.1109/IPDPS.2014.110>
- [34] “Intel® avx-512 - fp16 instruction set for intel® xeon® processor based products technology guide.” [Online]. Available: <https://networkbuilders.intel.com/solutionslibrary/intel-avx-512-fp16-instruction-set-for-intel-xeon-processor-based-products-technology-guide>
- [35] N. Nassif, A. O. Munch, C. L. Molnar, G. Pasdast, S. V. Lyer, Z. Yang, O. Mendoza, M. Huddart, S. Venkataraman, S. Kandula *et al.*, “Sapphire rapids: The next-generation intel xeon scalable processor,” in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 44–46.
- [36] “Wikidata.” [Online]. Available: <https://www.wikidata.org/>
- [37] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The case for learned index structures,” in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, G. Das, C. M. Jermaine, and P. A. Bernstein, Eds. ACM, 2018, pp. 489–504. [Online]. Available: <https://doi.org/10.1145/3183713.3196909>
- [38] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, “Approximate string joins in a database (almost) for free,” in *Proceedings of the 27th International Conference on Very Large Data Bases*, ser. VLDB ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, p. 491–500.
- [39] H. Zhang and Q. Zhang, “Minjoin: Efficient edit similarity joins via local hash minima,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’19. New York, NY, USA: Association for Computing