Numbers Matter! Bringing Quantity-awareness to Retrieval Systems

Satya Almasian, Milena Bruseva and **Michael Gertz**
Institute of Computer Science, Heidelberg University, Germany
{almasian,gertz}@informatik.uni-heidelberg.de
milena.k.bruseva@gmail.com

Abstract

Quantitative information plays a crucial role in understanding and interpreting the content of documents. Many user queries contain quantities and cannot be resolved without understanding their semantics, e.g., "car that costs less than \$10k". Yet, modern search engines apply the same ranking mechanisms for both words and quantities, overlooking magnitude and unit information. In this paper, we introduce two quantity-aware ranking techniques designed to rank both the quantity and textual content either jointly or independently. These techniques incorporate quantity information in available retrieval systems and can address queries with numerical conditions *equal*, *greater than*, and *less than*. To evaluate the effectiveness of our proposed models, we introduce two novel quantity-aware benchmark datasets in the domains of finance and medicine and compare our method against various lexical and neural models. The code and the benchmark datasets are available under https://github.com/satya77/QuantityAwareRankers.

1 Introduction

Despite advances in semantic search and sophisticated neural network architectures, handling quantitative information in text remains challenging. Specifically hard are quantity-centric queries in which the query contains a quantity and a numerical condition, e.g., "BMW with more than 530hp". The reason for this is that Information Retrieval (IR) systems are not aware of numbers and their semantics, such as proximity, in particular in combination with units. Numbers and units are treated in the same way as any other text token that is subject to subsequent processing, e.g., indexing or embedding. What complicates treating numbers and units in a proper way is that these objects can also have different surface forms (e.g., 6k vs 6,000 and mph vs miles per hour) and require standardization (Weikum, 2020). While there are approaches that specifically focus on numbers in text, e.g., extracting quantities for entities (Ho et al., 2019; Li et al., 2021), linking quantities in tables (Ibrahim et al., 2019), or numerical reasoning (Ran et al., 2019), they are tailored to specific tasks and not semantic search in general. This applies to neural models supporting IR, which are trained on general-purpose data without the focus on the quantity semantics. Language Models (LM), forming the basis for neural models, exhibit a limited understanding of number scales and proximity (Wallace et al., 2019). Despite recent work on numerical language models (Jin et al., 2021; Spokoyny et al., 2022), these architectures are very specific and require changes in the architecture of popularly used language models in IR, which indicates an expensive pre-training. Moreover, the lack of accessible quantity-centric benchmarks for training or comparing systems exacerbates the issue.

In this paper, we present two strategies to enhance the quantity understanding of current IR systems. We aim for a general-purpose model that is not specific to quantity ranking but is also capable of textual ranking. The two approaches differ in their integration of quantity ranking with textual ranking. The first employs a disjoint combination, while the second focuses on the joint ranking of quantities within the context of textual content. The disjoint approach is based on a complete separation of quantities from other textual tokens, from tokenization to the definition of relevance. It is an unsupervised and heuristic method, utilizing an index structure, compatible with various lexical and semantic IR models. On the other hand, in joint ranking, we aim to learn quantity-aware document and query representations through task-specific fine-tuning of neural IR models, without specific architectural changes. Further, we introduce two novel quantity-centric benchmarks, focusing on queries involving numerical conditions in the domains finance and medicine. We evaluate our proposed approaches

Findings of the Association for Computational Linguistics: EACL 2024, pages 12120–12136
November 12-16, 2024 ©2024 Association for Computational Linguistics

with various lexical and neural models and show significant improvements over the baselines.

2 Related Work

Related work for quantity-centric search is limited. (Ho et al., 2020, 2019) focus on quantity search for named entities, using a deep neural network for extracting quantity-centric tuples from text and query and matching based on context similarity. Their pipeline involves semantic role labeling and named entity extraction, both resource-intensive and reliant on sparsely available annotated data for quantities. Further, focusing on named entities limits the applicability to real-world scenarios.

QFinder (Almasian et al., 2022) integrates numerical and lexical indexes to enhance numerical understanding in a lexical IR system. Our disjoint model utilizes QFinder's heuristic ranking function, but we extend the QFinder ranker to include neural models and go beyond the limited query language, allowing users to provide queries in plain text. MQSearch (Maiya et al., 2015) extracts quantities with a set of regular expressions to create a rule-based system for finding documents containing certain keywords and ranges of values. Loosely related to IR, (Li et al., 2021) and (Rybinski et al., 2023) perform numerical summarization on unstructured text in the form of plots and graphs.

In the area of databases, there has been some work focusing on building numerical indices for queries that contain numerical restrictions (Agrawal and Srikant, 2003; Fontoura et al., 2007; Maiya et al., 2015). However, the main focus of such systems is the efficiency of the index structure and filtering out irrelevant numbers from the results with hard constraints rather than ranking.

Unlike quantity-aware IR, investigating numeracy in LMs is well-established. (Wallace et al., 2019) are among the first to highlight the limitations of embedding models when handling numbers. Subsequent efforts have led to dedicated embeddings and LMs for understanding scales, basic arithmetic, and numerical common sense knowledge (Jiang et al., 2020; Jin et al., 2021; Nogueira et al., 2021; Spithourakis and Riedel, 2018; Spokoyny et al., 2022; Sundararaman et al., 2020; Thawani et al., 2021). These LMs are specific to numeracy and not IR in general. While using them can enhance performance, we focus on improving quantity understanding in current IR models without architectural changes or training an LM from scratch.

3 Quantity-aware Model

A quantity-centric query contains a numerical condition, a value, and a unit, e.g., "iPhone XS with price under \$1500 ". Queries like "What is the price of iPhone XS?" are not considered quantity-centric as they do not require an understanding of scales and units. In the following, we assume a document collection where each document consists of a sequence of sentences. Following previous work (Almasian et al., 2022; Ho et al., 2019), we focus on sentences as retrieval units. A sentence $s_i := (T_i, Q_i)$ is a sequence of tokens $T_i = (t_1, ..., t_l)$ and quantities $Q_i = (q_1, ..., q_k)$, where a quantity $q_i = (u_i, v_i)$ is a tuple of a unit u_i and a value v_i. A quantity query is denoted by $X = (T_x, c, q_x)$, where $T_x = (t_{x_1}, .., t_{x_n})$ are the search terms related to the query quantity $q_x = (u_x, v_x)$. $c \in \{=, <, >\}$ represents a numerical condition, defining *equal*, *less than*, and *greater than* conditions. *Less than* and *greater than* indicate open bounds with values strictly less or greater than the query value. The *equal* condition pertains to values strictly equal to a query value. The relevance, $r(s_i|X)$, of sentence s_i to the query X is given in Eq 1. The similarity function sim_c is dependent on the query condition c, where τ is a generic function that maps a query and a document to their representations. Here, we explore different ways to define τ, which can be an embedding vector or a heuristic scoring function.

$$r(s_i|X) \sim sim_c(\tau(T_x, q_x), \tau(T_i, Q_i)) \quad (1)$$

In current IR systems, the notion of relevance in is based on textual similarity that does not account for values or units. Lexical retrieval systems prioritize word overlap, while semantic models focus on topic similarity. Neither approach is well-suited for handling numerical comparisons. For example, in the query "car with more than 320hp", a lexical system would search for the exact value "320" while a semantic model would focus on results related to car attributes, overlooking the quantity comparison. We need to alter the notion of relevance in these models in such a way that numerical conditions and unit matching plays a role in ranking.

We begin with a disjoint quantity-ranking method. Leveraging heuristic and supervised functions from (Almasian et al., 2022), we extend this approach to neural models. This model completely separates the ranking of quantities and textual element, such that a quantity proximity score can be added to

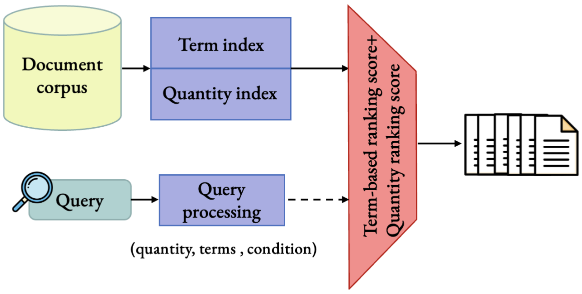


Figure 1: Disjoint quantity-ranking. A separate quantity index is used to compute quantity proximity and a term-based lexical or semantic index ranks the search terms.

a textual score. The independence of quantities and text has its shortcomings. To overcome those and also motivated by the learning capabilities of neural models, we propose a quantity-centric fine-tuning approach for neural IR systems. In the joint model, quantities and text are not treated separately, instead a neural model is fine-tuned to place an emphasis on numerical conditions during ranking.

3.1 Quantity Extraction

To facilitate both approaches, a prerequisite is a quantity extractor capable of identifying values (v), units (u), numerical conditions (c), and concepts (cn) associated with quantities. Concepts represent objects or events that numerical values refer to. For instance, in the sentence "The iPhone 11 has 64GB of storage", the concept is "iPhone 11 storage". For this purpose, we use the Comprehensive Quantity Extractor (CQE) framework (Almasian et al., 2023), a rule-based system capable of extracting all the mentioned components. However, this module can be substituted with any alternative well-performing quantity extractor.

3.2 Disjoint Quantity Ranking

The disjoint model is based on the separation of quantity and term ranking. We assume that the textual relevance of a sentence to query terms is independent of the proximity of query value and sentence values under the query condition. Then, the relevance of a sentence can be the summation of (1) textual similarity, and (2) quantity proximity under the query condition, as shown in Eq 2. Note that here, sim computes the similarity of search terms to a textual tokens of a sentence independent of sim_c, which computes the quantity proximities given query condition c. τ and $\tau^{'}$ signify that representations for query and document are not necessarily created from the same model. If the query is not quantity-centric, by removing the quantity score sim_c, the models fall back to term scoring.

$$r(s_i|X) \sim sim(\tau(T_x), \tau(T_i)) + sim_c(\tau^{'}(q_x), \tau^{'}(Q_i)) \tag{2}$$

In the following, we describe the computation of (1) term ($sim(\tau(T_x), \tau(T_i))$), and (2) quantity ($sim_c(\tau^{'}(q_x), \tau^{'}(Q_i))$ scorings. The general pipeline is illustrated in Fig. 1.

3.2.1 Quantity Scoring

Using a quantity index containing explicit information about values and units in normalized form, we use heuristic functions to compute the proximity of query and sentence values based on different numerical conditions.

Index creation: Documents are split into sentences that are processed independently by CQE, which outputs standardized values, e.g., \$300 million is converted to \$300,000,000, and normalized units, e.g., kilometer per hour and km/h are mapped to the same unit. A quantity index with unit/value pairs as keys is built from this output and resembles a lexical index. Each unique unit/value pair points to a list of sentences it occurs in.

Scoring functions: $sim_c(\tau(q_x), \tau(Q_i))$ is estimated by a scoring function qs that ranks the value in a sentence based on the value in the query given a numerical condition, where higher values indicate higher relevance. qs is dependent on the numerical condition c, resulting in different scores for the same values under different conditions. The quantity score only matters if the units match, otherwise, the values are not comparable and refer to different aspects of an object, e.g., the horsepower of a car is different from the km/h it reaches. qs is formulated in Eq 3. The indicator function $\mathbb{1}_{u_i}(u_x)$ enforces the match between the units of the query and the sentence, and Φ_c is the condition-dependent scoring function. To obtain a value between 0 and 1, the score is normalized by the number of quantities $|Q_i|$ in s_i. For brevity, from now on we refer to $qs(s, c, X)$ simply as qs.

$$qs(s_i, c, X) := \frac{1}{|Q_i|} \sum_{i=1}^{|Q_i|} \mathbb{1}_{u_i}(u_x)\Phi_c(v_x, v_i) \tag{3}$$

Φ_c refers to one of the three heuristic functions, one for each numerical condition (*equal*, *less than*, *greater than*), adapted from (Almasian et al., 2022). The study in (Almasian et al., 2022) explores various Φ functions and their implications for sorting of results (refer to App. A.1). Simply by changing Φ, results can be rearranged, independent of the training data. Nonetheless, for the evaluation of our model against other baselines, we focus only on the most intuitive variant, which ranks quantities with values closer to the query value in descending order. Φ for different conditions are shown in Eq 4. v_x is the query value, and v_i is the sentence value.

$$
\begin{aligned}
&\Phi_{=}(v_x, v_i) =: exp(-|v_x - v_i|) \\
&\Phi_{>}(v_x, v_i) =: \begin{cases} v_x/v_i & v_x{>}v_i \\ 0 & else \end{cases} \\
&\Phi_{<}(v_x, v_i) =: \begin{cases} v_i/v_x & v_x{<}v_i \\ 0 & else \end{cases}
\end{aligned}
\tag{4}
$$

$\Phi_{=}$ assesses the proximity of v_x to v_i by employing the exponential decay of their difference. The resulting score ranges between 0 and 1, with larger absolute differences yielding lower scores.

The scoring functions $\Phi_{<}$ and $\Phi_{>}$ determine numerical proximity based on the ratio of the query value v_x to the sentence value v_i, resulting in a score between 0 and 1. This ratio, independent of magnitude, yields higher scores for closer values.

3.2.2 Term Scoring

Term scoring, $sim(\tau(T_x), \tau(T_i))$, can come from any lexical or semantic ranker, requiring only normalized scores. Yet, IR systems typically do not normalize their scores, as it has no influence on the final ranking. Here, we discuss ways to normalize scores of lexical and semantic systems and combine them with qs. For a lexical model, we use BM25 (Robertson and Zaragoza, 2009), and for dense and sparse neural rankers, ColBERT (Khattab and Zaharia, 2020) and SPLADE (Formal et al., 2021) are employed. Other rankers can be employed following similar techniques.

Lexical model: Following (Almasian et al., 2022), we combine qs with the BM25 score. The combined score, represented in Eq 5, is constrained to sentences containing the search terms, as indicated by $\mathbb{1}_{T_x}(s_i)$. The parameter α controls the influence of the quantity scoring, falling back to pure term-based scoring when α is zero. The BM25(s_i, T_x) score is normalized per query by dividing each sentence's score by the maximum BM25 score retrieved for the specified search terms $\max_X = max_{s \in S}(\text{BM25}(s, T_x))$.

$$
\text{QBM25}(s_i, c, X) := \frac{\text{BM25}(s_i, T_x)}{\max_X} + \alpha \mathbb{1}_{T_x}(s_i) qs
\tag{5}
$$

Neural dense model: Representing a dense neural model, ColBERT is selected for the term scoring. This choice is due to the same model being used for joint quantity ranking, where token-level interactions are crucial. A contextualized term score is computed with the similarity computation between token embeddings of query and sentence, as in Eq 6. A ColBERT utilizes two BERT (Devlin et al., 2019) encoders for the query ($BERT(T_x)$) and document ($BERT(s_i)$) (sentence), where each encoder outputs a list of token embeddings.

$$
\text{ColBERT}(s_i, T_x) = \sum_{k \in [|\text{BERT}(T_x)|]} max_{j \in [|\text{BERT}|(s_i)|]} \text{BERT}(T_x)_k \cdot \text{BERT}(s_i)_j
\tag{6}
$$

A ColBERT score comes from the MaxSim operation between the token embeddings of query and sentence. MaxSim calculates an unbounded score for the maximum cosine similarity between the token embeddings. To normalize this score, we need the maximum score. However, calculating the maximum score for the entire collection is impractical. For ranking, ColBERT leverages the pruning-friendly nature of the MaxSim in an approximate nearest neighbor search (Johnson et al., 2019) to return the top-k most relevant candidate sentences S_k. We compute the maximum score based on these candidate sentences $max_X = max_{s \in S_k}(\text{ColBERT})$ to normalize the score between 0 and 1. qs is then exclusively applied to the top-k candidates, serving as a second-stage re-ranker for numerical proximity. The final score is defined in Eq 7, where α again controls the impact of quantity scoring.

$$
\text{QColBERT}(s_i, c, X) := \frac{\text{ColBERT}(s_i, T_x)}{\max_X} + \alpha \cdot qs
\tag{7}
$$

Note that qs only affects the top-k sentences. We also present a neural sparse model, where qs is integrated into the entire ranking.

Neural sparse model: The SPLADE model extends the document and query terms and uses an inverted index for sparse dot products, allowing for end-to-end integration with the quantity scoring. Inside the index, instead of term frequencies, term importance weights are computed by SPLADE. For each sentence and query, the BERT embeddings are passed through a ReLU non-linearity and log function to produce a sparse vector over the entire vocabulary, where the values of this vector are the term importance scores. Then the relevance of the query to a sentence is based on the sparse dot product of this vector, as shown in Eq 8.

$$\text{SPLADE}(s_i, T_x) := \log(\mathbf{1} + \text{ReLU}(\text{BERT}(s_i))) \cdot \log(\mathbf{1} + \text{ReLU}(\text{BERT}(T_x))) \tag{8}$$

We normalize the SPLADE score by the maximum score for a given query, $\max_X = max_{s_i \in S}(\text{SPLADE}(s_i, T_x))$, as defined in Eq 9. For higher precision, the quantity score is only added to sentences where there is a match between the expanded query terms and documents, denoted by the indicator function $\mathbb{1}$.

$$\text{QSPLADE}(s_i, c, X) := \frac{\text{SPLADE}(s_i, T_x)}{\max_X} + \alpha \mathbb{1}(s_i) qs \tag{9}$$

3.3 Joint Textual and Quantity Ranking

The independence assumption between quantities and terms allows to easily combine a quantity score with various textual rankers. The heuristic nature of quantity ranking functions allows for an easy alteration of the notion of relevance, without any fine tuning. However, it can also be problematic. Consider the query "iPhone XR below €200". In a disjoint ranking, the following sentences can receive an inappropriately high score.

1) *The price of an iPhone XR reached €236.50, whereas Samsung A14 is €132.00.* This sentence has multiple quantities. The numerical condition is satisfied for a value unrelated to "iPhone XR".

2) *Older iPhones, including iPhone XR have dropped in price with iPhone 8 to €152.94.* Here, "iPhone XR" has no associated quantity.

These cases are due to a lack of correct association between concept and quantity. We refer to this as *quantity-concept mismatch*. To address this, we need to rank sentences based on quantities in context. Transformer-based models inherently capture token inter-dependencies across the entire context. However, current benchmarks lack quantity-centric data. Therefore, it remains unclear whether the deficiency in quantity understanding is due to the absence of task-specific training data or if the current architectures hinder numerical comparisons altogether. To investigate this, we propose a data generation approach to create synthetic quantity-centric queries and positive and negative samples. The data is created focusing on two common problems of neural models.

First is the inability to perform value comparisons given numerical conditions. For the query "iPhone XR below €200", neural models often ignore the *less than* (below) condition and focus on the semantic or topic similarity of query text and sentence. Second, the semantic similarity of units is not well-defined. Therefore, results with "dollar" and other currencies receive high scores due to the context similarity of the currency units. Please see App. B.6 for a more detailed discussion.

Our data generation paradigm is designed to enhance *value comparisons* and understanding of *unit surface forms*, by generating contrastive positive and negative sentences through data augmentation. Data augmentation, widely used in computer vision, has also found applications in NLP tasks (Sennrich et al., 2016). The GENBERT model (Geva et al., 2020) is a relevant example, which employs templates for generating pre-training data to enhance numerical reasoning in question-answering systems without specialized architectures.

Similar to GENBERT, we fine-tune neural IR models, ColBERT and SPLADE, on synthetic data for quantity-centric IR [1]. Three stages of data generation is described in the following: *quantity extraction*, *query generation*, and *sample generation*.

3.3.1 Quantity Extraction

The documents are split into sentences and fed to CQE to extract quantities and concepts. The corpus is then transformed into an index-like structure based on concepts and units. We refer to this structure as *concept/unit index*. The keys of the index are concept/unit pairs that point to a list of values associated with the pair and a list of respective sentences they occur in. The list of values can be viewed as the distribution of values for a concept under a specific unit. An example entry is shown in App. B.1. We utilize this index structure in the subsequent steps for query and sample generation.

3.3.2 Query Generation

For each concept/unit pair, three queries, one for each condition, are created with the template

```
query = {concept} {numerical_condition}
{unit_before}{value}{unit_after}.
```

The variables enclosed in the brackets are populated during query generation. These steps are shown in the algorithm in App. B.2. Here, we describe how each placeholder is filled.

Unit: A surface form of the query unit is chosen randomly from a dictionary of unit surfaces provided by CQE, e.g., "€" is a surface of the unit

[1] Given that we are perturbing values and units in a sentence, one might alternatively call this *data perturbation*.

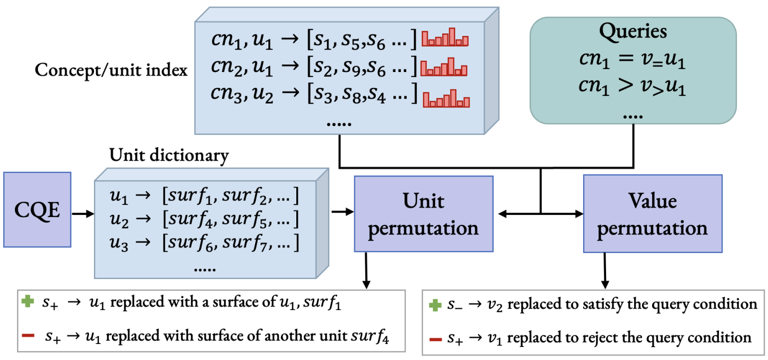


Figure 2: Sample generation. The input are the queries from the query generation step, *concept/unit index* and the unit dictionary from CQE. Additional positive and negative samples are generated using value and unit permutation.

"euro". `unit_before` and `unit_after` account for symbols appearing before, e.g., "€" and abbreviations after a value, e.g., "EUR".

Value: For sample generation, sentences containing values meeting the query condition are crucial. Therefore, selecting query values with enough supporting sentences is vital. We propose the following strategy, based on the value distribution in the *concept/unit index* for each concept/unit pair:

Equal query: Query values are chosen from the most frequent values in the index (peak of value distributions), ensuring the availability of maximum supporting sentences for a given concept/unit.

Less and greater than queries: For these bounds, optimal candidates are close to the mean of the value distribution, such that when the numerical condition is applied more sentences fall within limits. Infrequent values (tail of the distribution) may have inadequate supporting sentences for the sample generation step. Refer to the App. B.3 for examples of the value selection.

To avoid systemic bias by focusing on the most frequent values, we generate a second set of queries for each concept/unit pair by picking the query values at random for each condition.

To account for variability in representation, surface forms of large values that have multiple written forms are randomly replaced with their written form. This takes the shape of a composite of numbers and postfixes, such as "10 million," or includes commas for digit separation, e.g., "10,000,000".

Numerical Condition: This is a phrase in natural language indicating a bound on a quantity, e.g., "above" for a *greater than* condition. To this end, a surface-form dictionary is created, and the respective placeholder is filled with values randomly chosen from the dictionary (see App. B.4).

Concept: CQE identifies multi-word spans in a sentence as concepts. Utilizing them directly for query generation overlooks the nuances of semantic queries. For example, in the sentence "Disney+ charges $6.99 a month.", "Disney+" is the extracted concept. "Disney+" is a streaming platform, including other media services. Such a sentence is relevant for a lexical query with exact matches, e.g., "Disney+ price under $7.99 a month", or for a semantic query, e.g., "streaming platform price over 5 dollar/month". Relying exclusively on keywords in sentences poses the risk of biasing the neural models toward lexical search and away from semantic search. To avoid such a case, we add *concept expansion*, where a large language model, such as GPT-3 (Chen et al., 2023), is used to generate synonyms or synsets for a given concept (see App. B.5). These expansions are used to generate semantic queries. E.g., "Disney+" becomes "Streaming platform". For each expanded concept, new values and unit surface forms are sampled to generate semantic queries.

3.3.3 Sample Generation

The input of this stage are the generated queries and the *concept/unit index*. The sample generation step creates positive and negative training samples for each quantity-centric query. This includes positive and negative samples obtained directly from the dataset as well as additional augmented samples. The sample generation pipeline is shown in Fig. 2. See App. B.7 for algorithmic overview of the steps. Here, we describe each step in detail.

Look-up: Given a query containing a *(concept, unit, condition, value)*, a lookup in the *concept/unit index* is performed to retrieve the sentences and the distribution of values.

Positive and negative sentences: The obtained sentences from *concept/unit index* are divided into positive s_+ and negative s_- lists, based on the numerical condition. s_+ contains sentences where the values in them satisfy the query condition, and s_- contains sentences violating the condition.

Original sampling: With sample size n, sentences are randomly selected from s_+ as positive samples (s_{o+}) and from s_- as negative samples (s_{o-}). Refer to App. B.9 for more information on sampling.

Unit permutation sampling: This method generates positive and negative samples to cover diverse unit surface forms using CQE's unit dictionary. Positive samples contain various surface forms of the unit in the query (correct unit), while negative samples include surface forms of units in the same family as the query unit (incorrect unit in similar context), creating negatives.

- A positive sample, s_{u+}, is created by substituting the unit in a positive sentence, s_+, with other surface forms of the unit in query u_x.

- A negative sample, s_{u-}, is created by replacing the unit in a positive sentence, s_+, with a surface form of a unit different from the query unit,u_x, but belonging to the same family. The unit families are grouped based on the property they measure. For example, "pace", "meter", and "foot" all belong to the family of "length". Sampling the surface form from the same family ensures a fine distinction between unit types, even in similar contexts.

Value permutation sampling: This permutation emphasizes the importance of the value comparison and numerical conditions, highlighting that sentence relevance depends on whether the sentence value satisfies the query condition or not.

- A positive sample, s_{v+}, is created by permuting the values in a negative sentence s_-, maintaining the correct concept and unit but adjusting the value to satisfy the quantity condition.

- A negative sample, s_{v-}, is generated by permuting the values in a positive sentence s_+, where concept, unit, and value are all correct, to invalidate the quantity condition.

The replacement values are sampled from the values in the *concept/unit index*, mirroring the underlying distribution of the relevant quantity, for to the reason for this choice, refer to App. B.8.

Table 1: Query types in FinQuant and MedQuant.

	FinQuant	MedQuant
Total queries	420	210
Sentences	306,291	153,252
Sentence > one quantity	42,633	53,668
Per condition	140	70
Keyword-based queries	300	120
Semantic queries	120	90

Aggregate: The final set of positive and negative samples for each query is the union of all samples generated from the original sampling, value and unit permutation, $s_{f+} = s_{o+} \cup s_{u+} \cup s_{v+}$ and $s_{f-} = s_{o-} \cup s_{u-} \cup s_{v-}$.

4 Evaluation

Given the absence of task-specific models, we assess our quantity-aware models against general domain lexical and neural models.

Lexical models include a BM25 and a BM25$_{filter}$ variant. BM25$_{filter}$ has a separate numerical index to eliminate the results of BM25 where the query condition is not met. This method resembles numerical indices from databases as it works by filtering rather than by ranking.

Neural models include the trained checkpoints of SPLADE and ColBERT as well as Cohere$_{v3}$ [2]. Cohere$_{v3}$ shows that even industry-level models trained on extensive data lack quantity awareness.

4.1 Datasets

We introduce two English resources called FinQuant and MedQuant. To the best of our knowledge, these are the first quantity-centric benchmarks for retrieval. Test queries were manually formulated using the concept/unit index, covering both lexical and semantic queries. Statistics for various query types are presented in Table 1. There is an equal number of queries for each condition, and semantic queries constitute a smaller portion due to annotation challenges. For details on the dataset creation, see App. C.1. The data is annotated by the first two authors of the paper, with inter-annotator agreement computed on a subset of 20 samples per dataset. The Cohen's Kappa coefficient (Cohen, 1960) is 0.83 and 0.88 for FinQuant and MedQuant, respectively. The FinQuant corpus contains over 300k sentences from 473,375 news articles. MedQuant is smaller, containing over 150k sentences from 375,580 medical documents of the

[2] https://cohere.com/embeddings DLA: 27.05.24

Table 2: P@10, MRR@10, NDCG@10 and R@100 on FinQuant and MedQuant. The top-2 values in each column (for each metric) are highlighted in bold.

	Model	latency (ms)	FinQuant				MedQuant			
			P@10	MRR@10	NDCG@10	R@100	P@10	MRR@10	NDCG@10	R@100
baselines	BM25	9	0.06	0.14	0.09	0.47	0.04	0.11	0.07	0.37
	BM25$_{filter}$	9	0.14	0.32	0.25	0.60	0.08	0.19	0.15	0.48
	Cohere$_{v3}$	-	0.14	0.22	0.19	0.27	0.10	0.17	0.15	0.25
	SPLADE	26	0.10	0.24	0.19	0.53	0.11	0.25	0.20	0.58
	ColBERT	36	0.15	0.35	0.27	0.70	0.12	0.31	0.24	0.63
disjoint	QBM25	311	0.21	0.53	0.41	0.55	**0.18**	0.47	**0.37**	0.51
	QSPLADE	319	**0.29**†	**0.67**†	**0.53**†	**0.83**†	**0.19**†	**0.52**†	**0.38**†	0.70†
	QColBERT	42	**0.30**†	**0.69**†	**0.56**†	**0.87**†	**0.18**†	**0.51**†	**0.37**†	**0.73**†
joint	SPLADE$_{ft}$	26	0.21†	0.51†	0.41†	0.74†	0.14†	0.37†	0.29†	0.63†
	ColBERT$_{ft}$	36†	0.23†	0.55†	0.44†	0.77†	0.18†	0.44†	0.36†	**0.72**†
cross-dataset	SPLADE$_{out}$	26	-0.03	-0.06	-0.07	-0.04	-0.02	-0.01	-0.04	-0.05
	ColBERT$_{out}$	36	-0.03	-0.07	-0.06	-0.03	-0.02	-0.01	-0.03	-0.02

TREC Medical Records (Voorhees, 2013). Since the concept/unit index is used for dataset creation, CQE's performance directly affects the data quality. While CQE is adept at handling financial data, extractions on clinical data were noisy. However, we find it important to report results on both datasets, making the reader aware of the lower quality of MedQuant.

4.2 Ranking Performance

Table 2 shows the ranking performance of quantity-aware models, in terms of P@10, MRR@10, NDCG@10, R@100, and latency in milliseconds. The three models with a "Q" prefix indicate the disjoint and unsupervised rankers. Neural models with a $_{ft}$ postfix are joint models fine-tuned on synthetic data. Permutation re-sampling is used to test for significant improvements (Riezler and Maxwell, 2005). Results denoted with † mark highly significant improvements over the baseline models, without quantity awareness with a p-value < 0.01. All results are from single runs. For implementation details refer to App. C.3.

Contrary to our initial hypothesis, disjoint rankers consistently outperform joint models across all metrics, with improvements exceeding 10 points in P@10 and over 30 points in MRR and NDCG over the base models (without the "Q" prefix), without requiring additional fine-tuning. The only drawback of the disjoint models is a minimal increase in latency, especially for QBM25 and QSPLADE, where the quantity score is added to the entire ranking. This overhead diminishes for the top-performing model, QColBERT, where the quantity score serves as a re-ranker on the top-k candidates. ColBERT shows high recall on both datasets, suggesting that relevant results are within the top-k but not necessarily at the very top. Hence, re-ranking with a quantity score proves beneficial.

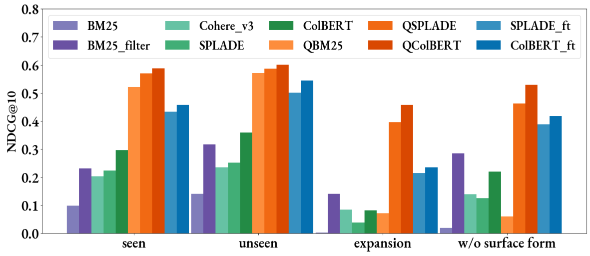
The joint models show a comparable performance boost, with metrics falling below those of the disjoint ranker but still improving from the base models. This validates our hypothesis that the absence of task-specific data has amplified the challenge of quantity understanding for retrieval systems. Here, once again the ColBERT$_{ft}$ variant shows superior performance. We attribute the better performance of the ColBERT-based model to the fine-grained token-level interactions that allow the model to learn better associations between tokens. In quantity ranking, token interactions play a more significant role compared to the query and document expansions conducted by SPLADE. This also showcases that the architecture and how the inter-token interactions are modeled matter for quantity understanding. Nonetheless, even after fine-tuning, understanding numerical conditions remains a challenge. We investigate how much the fine-tuned models rely on quantities for ranking in App. C.4.

4.3 Cross-dataset Generalization

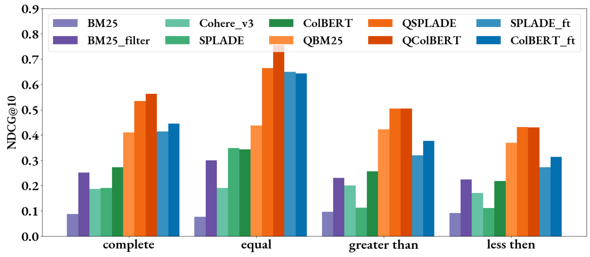
The two lower bottom rows of Table 2 list the performance drop of joint rankers on out-of-domain data, compared to models fine-tuned on generated data from the same domain. Each model is fine-tuned on data from the other dataset and shows a minimal performance drop, suggesting that the models learn patterns for quantity-centric queries without memorizing common queries.

4.4 Lexical vs Semantic Queries

Fig. 3a shows NDCG@10 of all models on lexical and semantic subsets of the FinQuant. The *seen* and *unseen* are lexical queries and *expansion* and

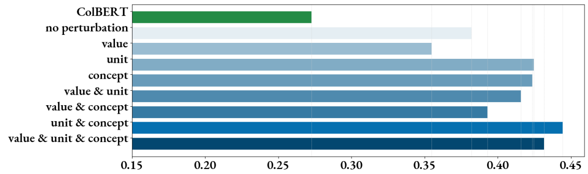


(a) Lexical vs semantic subsets

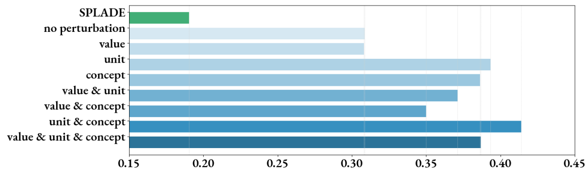


(b) Subset with different numerical conditions

Figure 3: Performance on different subsets of FinQuant.



(a) ColBERT_{ft}



(b) SPLADE_{ft}

Figure 4: Ablation study of different augmentation methods, where *value* and *unit*, refer to value and unit permutation and *concept* refers to concept expansion.

w/o surface form represent semantic queries. For the details of their differences, see App. C.2. Interestingly, the disjoint ranking using dense models captures both semantic similarity and quantity understanding. QBM25 performs equally well for lexical queries but significantly worse on semantic ones. Joint rankers outperform base models, without quantity-awareness, in both lexical and semantic queries but lag behind disjoint models.

Fig. 3b depicts NDCG@10 of all models on different numerical conditions. *Equal* queries are in general easier for the models as the notion of relevance in this case aligns with textual ranking. The performance drops almost 20 points for the bound-based conditions. This drop is consistent across all models, implying that the bound-based conditions are harder for models to rank.

4.5 Ablation Study on Augmentation Methods

We conduct an ablation study to evaluate the impact of various augmentation strategies. The results for ColBERT_{ft} and SPLADE_{ft} on the FinQuant dataset are shown in Figures 4a and 4b, respectively. *No perturbation* refers to training with only the original positive and negative samples. Surprisingly, value permutation has a negative impact when combined with other augmentation methods. Therefore, the best results for both SPLADE and ColBERT models come from unit permutation and concept expansion. This behaviour can be related to the internal representation of the neural models, hindering their ability to correctly learn magnitude comparisons.

5 Conclusion and Ongoing Work

In this work, we (1) examined the shortcomings of prominent IR models concerning quantity-centric queries, (2) proposed two methods, joint and disjoint quantity-aware models, to integrate quantity understanding into both classical IR models as well recent neural architectures, (3) introduced two novel benchmark datasets containing quantity-centric queries, which to our knowledge are the first of their kind, and (4) demonstrated significant improvements in quantity understanding over the baselines for both proposed techniques.

Our joint and disjoint approaches enable the integration of quantity understanding without altering existing architectures and with minimal overhead in query latency. We further highlight the strengths and weaknesses of both approaches, arguing that the choice of method should depend on the specific use case scenario. The disjoint approach, which relies on a quantity index for ranking, consistently outperforms joint models across various domains. Its unsupervised and heuristic nature also makes it more flexible than the joint rankers. However, despite the lower performance, the joint approach eliminates the need for an external index and the associated latency increase. Due to the lack of existing quantity-aware IR models, most of our baselines are general-purpose, but we hope that our systems can serve as baseline for future work in this direction and that our benchmarks encourage the researchers to work on this task. In the future, we plan to explore the impact of dedicated numerical embeddings and LMs in information retrieval.

6 Limitations

In this section, we highlight the limitations of the proposed evaluation resources and the models introduced in this paper.

Evaluation resources: One immediate consideration regarding the datasets is the relatively limited number of test queries compared to larger-scale datasets such as MSMARCO (Nguyen et al., 2016). This is mainly due to limited human resources and budget in an academic setting. Nonetheless, we argue that this number of queries is already enough to showcase certain quantity-centric capabilities. Another shortcoming of the data is the absence of queries for *ranges*, e.g., "iPhone with price between 500 and 800 dollars", and *negations*, "iPhones not equal to 500 dollars". In our future work, we will to introduce benchmarks that cover these cases as well.

Quantity-aware models: When considering neural models, one limitation is their reliance on hardware capabilities, particularly the need for GPUs to ensure efficient training, indexing, and inference. The query latency values reported in this paper would suffer greatly if the computations were done only on a CPU.
Both the synthetic data generation paradigm and the disjoint model rely on a quantity extractor. In the case of the disjoint model, the quality of the quantity index directly relies on the quality of value and unit extraction. If a value and unit are not detected by the extractor they will not be considered by the scoring function. In the joint model, for data generation, the quantity extractor should also possess the ability to detect concepts in text, introducing the potential for additional error propagation through the system. The performance of the quantity extractor used in this study (CQE) is not discussed here, as it is covered in detail in its respective paper (Almasian et al., 2023). As for the impact of false extractions, this cannot be directly quantified because it would require a dataset with a gold standard not only for relevant passages but also for quantity extractions, which is not available. Furthermore, in the case of neural models, the textual and quantity ranking are intertwined, making it difficult to identify the source of the error. Nevertheless, the improvements over baseline models demonstrate that even with an imperfect quantity extractor, enhancements can still be made to existing systems.
In this work, we do not discuss models for *ranges* and *negations*. Such variations to the disjoint models requires only a change in the numerical scoring function but it is more difficult for the joint setting, where proper training data is required.
For the bound-based conditions of *less than* and *greater than*, we considered open bounds. Depending on the user intent, closed bounds might be more appropriate, however, similar to the optimal sorting of results, this issue does not have a single solution.

References

Rakesh Agrawal and Ramakrishnan Srikant. 2003. Searching with Numbers. *IEEE Trans. Knowl. Data Eng.*, 15(4):855–870.

Satya Almasian, Milena Bruseva, and Michael Gertz. 2022. QFinder: A Framework for Quantity-centric Ranking. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3272–3277. ACM.

Satya Almasian, Vivian Kazakova, Philip Göldner, and Michael Gertz. 2023. CQE: A Comprehensive Quantity Extractor. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 12845–12859. ACL.

Zekai Chen, Mariann Micsinai Balan, and Kevin Brown. 2023. Language Models are Few-shot Learners for Prognostic Prediction. *CoRR*, abs/2302.12692.

Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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*, pages 4171–4186. ACL.

Marcus Fontoura, Ronny Lempel, Runping Qi, and Jason Y. Zien. 2007. Inverted Index Support for Numeric Search. *Internet Math.*, 3(2):153–185.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2288–2292. ACM.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting Numerical Reasoning Skills into Language Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL*, pages 946–958. ACL.

Vinh Thinh Ho, Yusra Ibrahim, Koninika Pal, Klaus Berberich, and Gerhard Weikum. 2019. Qsearch: Answering Quantity Queries from Text. In *The Semantic Web - ISWC - 18th International Semantic Web Conference, Proceedings*, volume 11778 of *Lecture Notes in Computer Science*, pages 237–257. Springer.

Vinh Thinh Ho, Koninika Pal, Niko Kleer, Klaus Berberich, and Gerhard Weikum. 2020. Entities with Quantities: Extraction, Search, and Ranking. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining*, pages 833–836. ACM.

Yusra Ibrahim, Mirek Riedewald, Gerhard Weikum, and Demetrios Zeinalipour-Yazti. 2019. Bridging Quantities in Tables and Text. In *35th IEEE International Conference on Data Engineering, ICDE*, pages 1010–1021. IEEE.

Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. 2020. Learning Numeral Embedding. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 2586–2599. ACL.

Zhihua Jin, Xin Jiang, Xingbo Wang, Qun Liu, Yong Wang, Xiaozhe Ren, and Huamin Qu. 2021. NumGPT: Improving Numeracy Ability of Generative Pre-trained Models. *CoRR*, abs/2109.03137.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48. ACM.

Tongliang Li, Lei Fang, Jian-Guang Lou, Zhoujun Li, and Dongmei Zhang. 2021. AnaSearch: Extract, Retrieve and Visualize Structured Results from Unstructured Text for Analytical Queries. In *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining*, pages 906–909. ACM.

Arun S. Maiya, Dale Visser, and Andrew Wan. 2015. Mining Measured Information from Text. In *Proceedings of the 38th International SIGIR Conference on Research and Development in Information Retrieval*, pages 899–902. ACM.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS)*, volume 1773 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Rodrigo Frassetto Nogueira, Zhiying Jiang, and Jimmy Lin. 2021. Investigating the Limitations of the Transformers with Simple Arithmetic Tasks. *CoRR*, abs/2102.13019.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. NumNet: Machine Reading Comprehension with Numerical Reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing EMNLP-IJCNLP*, pages 2474–2484. ACL.

Stefan Riezler and John T. Maxwell. 2005. On some pitfalls in automatic evaluation and significance testing for MT. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 57–64, Ann Arbor, Michigan. ACL.

Stephen Robertson and Hugo Zaragoza. 2009. *The Probabilistic Relevance Framework: BM25 and Beyond*. Now Publishers Inc.

Maciej Rybinski, Stephen Wan, Sarvnaz Karimi, Cécile Paris, Brian Jin, Neil I. Huth, Peter J. Thorburn, and Dean P. Holzworth. 2023. SciHarvester: Searching Scientific Documents for Numerical Values. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR*, pages 3135–3139. ACM.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving Neural Machine Translation Models with Monolingual Data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL*. ACL.

Georgios P. Spithourakis and Sebastian Riedel. 2018. Numeracy for Language Models: Evaluating and Improving their Ability to Predict Numbers. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2104–2115. ACL.

Daniel Spokoyny, Ivan Lee, Zhao Jin, and Taylor Berg-Kirkpatrick. 2022. Masked Measurement Prediction: Learning to Jointly Predict Quantities and Units from Textual Context. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 17–29. ACL.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. 2020. Methods for Numeracy-Preserving Word Embeddings. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 4742–4753. ACL.

Avijit Thawani, Jay Pujara, Filip Ilievski, and Pedro A. Szekely. 2021. Representing Numbers in NLP: a Survey and a Vision. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pages 644–656. ACL.

Christophe Van Gysel and Maarten de Rijke. 2018. Pytrec_eval: An Extremely Fast Python Interface to trec_eval. In *SIGIR*. ACM.

Ellen M. Voorhees. 2013. The TREC Medical Records Track. In *ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics. ACM-BCB 2013*, page 239. ACM.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP Models Know Numbers? Probing Numeracy in Embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP*, pages 5306–5314. ACL.

Gerhard Weikum. 2020. Entities with Quantities. *IEEE Data Eng. Bull.*, 43(1):4–8.

A Disjoint Quantity Ranking

In this section, we provide additional material related to the disjoint quantity ranking model.

A.1 Optimal Sorting

Although all the sentences that satisfy a query condition and have the correct concept and unit are potentially relevant to a given query, the order in which the result items are presented to the user can either aid or hinder the user in finding the desired result. In term-based ranking, the optimal order of results is evident. However, when it comes to quantities, relevance is more subjective and the optimal sorting is dependent on the user's information needs. For example, a user searching for "iPhone camera that has more than 8 inches" might look for a maximum value larger than 8 inches or a display only marginally larger, both of which are valid answers. Presenting results in ascending or descending order based on numerical distances allows the user to identify the desired result more efficiently. (Almasian et al., 2022) briefly addresses this issue and explores potential alternatives for scoring functions to enable various sorting options.

Our disjoint approach is flexible concerning different sorting orders. By switching a scoring function, the results can be rearranged. The joint models proposed here are not as adaptive, and rearranging the results requires additional fine-tuning based on a new preferred sorting.

B Joint Quantity Ranking

In this section, we provide additional information related to the joint quantity ranking model.

B.1 Concept-unit Index

An example entry in the *concept/unit index* from the FinQuant dataset is shown below.

```
{("cannabis company", "cent per share"):
{"values":[0.9, 1.4, 17.0, 17.0, 22.0,
26.0, 35.0, 84.0],
"sentences":['The cannabis company says
 the loss amounted to 0.9 of a cent per
 share for the quarter ended May 31
 compared with a loss of $4 million or
 1.4 cents per share a year earlier .',
'The cannabis company says its loss
amounted to 17 cents per diluted share
 for the quarter ended Jan. 31 .',...]}}
```

Note that the repetition of values for the same concept/unit pair is stored as duplicates, such that the frequency of values is kept for the distribution, e.g., the value "17.0" is repeated twice as it occurs in two sentences.

The index creation steps are depicted in Fig. 5. The corpus is processed with CQE to extract values, units, and concepts from each sentence. The sentences sharing the same unit and concept are grouped into a list, along with values in each sentence. The list of values represents a frequency distribution of concept for a certain unit.

B.2 Query Generation

The complete query generation pipeline is depicted in Fig. 6. The *concept/unit index* is used to select values and units for numerical conditions. Additionally, a large LM (in our case GPT-3) is used to expand concepts for semantic queries. The template generation block combines all the outputs of other blocks to formulate three queries for each concept/unit pair. To generate queries for expanded concepts, a new query value is chosen from the associated value distribution, and a new set of queries is formulated.

We offer the query generation pseudocode in Algorithm 1 to make the input and output of each step clear. In the algorithm, v refers to a value, u to a unit, c to a condition, and cn to a concept.

It is worth noting that although we used GPT-3 for concept expansion, the method is independent of the LM used and this part of the pipline can be replaced with the LM of choice.

B.3 Choosing the Right Query Value

Each entry in the *concept/unit index* points to the list of sentences conating the concept and unit pair and list of values in those sentences. For the

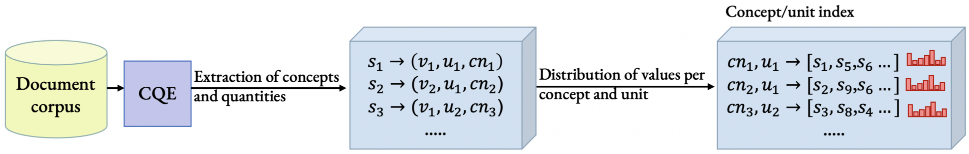


Figure 5: Overview of the quantity tagging step and creation of *concept/unit index* structure. The index contains distinct unit and value pairs, which point to sentences they occur in and a distribution of values.

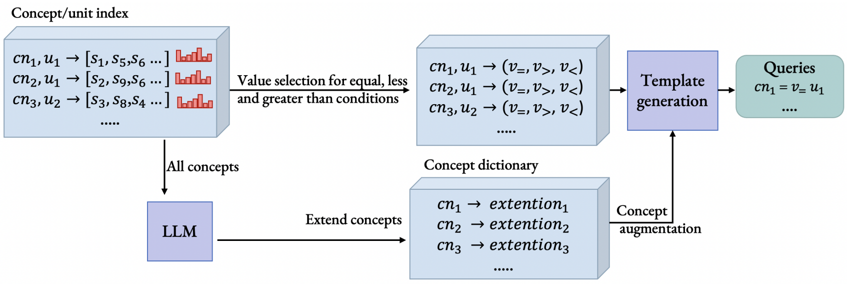


Figure 6: Overview of the query generation pipeline, using *concept/unit* index and a large LM for concept expansion. The expanded concept are saved in a dictionary and used alongside the *concept/unit* index in template generation. For each concept and unit, three queries are generated, one for each numerical condition.

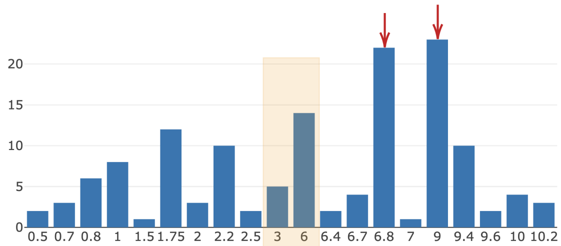


Figure 7: An example of choosing query values from a value distribution for *equal* condition (marked with red arrows) and bound-based conditions (marked with a yellow box).

Algorithm 1 Query Generation

function GENERATE_QUERY(cn, u, c)
  $v \leftarrow$ get_query_values(cn_unit_dict, c)
  $u\_b, u\_a \leftarrow$ get_unit_surfaceform(u)
  $c \leftarrow$ get_condition_surfaceforms(c)
  $query \leftarrow conc + c + u\_b + v + u\_a$
  return query
end function

cn_unit_dict $\leftarrow$ concept/unit index
cn_expand_dict $\leftarrow$ concept expansions
for (cn, u) in cn_unit_dict **do**:
  for cn in $[cn$, cn_expand_dict$[cn]]$ **do**:
    for c in $(equal, greater, less)$ **do**:
    GENEARATE_QUERY(cn, u, c)
    end for
  end for
end for

data augmentation, we require a number of positive and negative samples per query and therefore, it is important to choose the value of the query such that supporting sentences in the corpus are present. A hypothetical example of value distribution is shown in Fig. 7. For the *equal* query, the challenge is to find enough positive samples, since there is an abundance of values not equal to the query value in each distribution. In Fig. 7, values with the highest frequency, denoted by red arrows pointing to peaks in the distribution, serve as optimal candidates for the equal condition. In this manner, we make sure that there are enough positive samples for the data augmentation. Values close to the average (highlighted in a yellow box) are chosen for the *less than* and *greater than* queries. For such queries, we avoid infrequent values towards the tail of the distribution, to circumvent too few positive or negative samples.

B.4 Dictionary of Numerical Conditions

A non-comprehensive dictionary of surface forms for numerical conditions is shown in Table 3, containing multiple surface forms for each condition.

Table 3: Numerical conditions used for query generation and their surface forms.

Condition	Surface forms
Equal	exactly, exact, equals, equals to, for, with, of, at
greater than	greater than, more than, above, larger than, over, higher than, exceed, exceeding
Less than	smaller than, below, less than, fewer than, no more than, beneath

B.5 Concept Expansion

For concept expansion, we use the OpenAI API [3] and employ the `text-davinci-003` model with few-shot learning. We set the temperature to 1 to encourage creative responses. Since the concepts come from two distinct domains of finance and medicine, the few-shot examples vary accordingly. Below we specify the two prompts used for concept expansion, the result is stored in a concept expansion dictionary and utilized during query generation. The placeholder `{concept}` is replaced with a concept from the *concept/unit index* that is meant to be expanded.

For the financial domain:

```
Complete with the words super set or
synonym, but do not reuse the exact
same words, the word "Super Set"
should not be in the response and
response should have at least two words:

S&P 500 = stock market index
Audi = car
Oil prices = petroleum prices
unemployment rate = unemployment
percentage
iPhone sales = phone sales
Netflix shares = stock shares
President Trump = President
iPhone 11= iphone
Hong Kong = city
stake PEXA = Property Exchange Australia
shares

{concept} =
```

For the medical domain:

```
Complete with the words super set or
synonym, but do not reuse the exact
same words, the word "Super Set"
should not be in the response and
response should have at least two words:

ophthalmic solution = eye medication
Control group = treatment group
irinotecan hydrochloride = chemotherapy
drug
monoclonal antibody = substitute
antibodies
MRI scans = Magnetic resonance imaging
influenza H1N1 vaccine = flu vaccine
HAI antibody response = Influenza-specific
antibody response

{concept} =
```

[3] https://openai.com/ DLA:27.05.2024

B.6 When Semantic Search Backfires

Semantic retrieval systems consider an entire context to determine relevance to a query at hand. Often, this aligns well with the user's expectations. For instance, when searching for a "dark color evening dress", any dress that can be worn as an evening gown and has a dark color would be suitable. But as soon as the user becomes more specific like "blue evening dress", the embedding space could also bring a similar color like "teal" into the search result. Depending on the user's flexibility regarding the dress color, this behavior may or may not be desirable. Such hard constraints are challenging for neural models. Quantity-centric queries impose hard constraints on values and units where the fuzzy matching of context might do more harm than good. For instance, when searching for a "car with more than 320 hp", if the results contain a car with "360 brake horsepower" instead of horsepower, the result is irrelevant. Both horsepower and brake horsepower are used in similar contexts but refer to different attributes. Horsepower measures the power generated by the engine, while brake horsepower measures how much of the power produced by the engine is sent to the wheels that make the car accelerate. Another common problem is with currencies. Given that monetary values often appear in similar contexts, it becomes challenging for neural models to differentiate between various currency units. The same applies to hard constraints on values, where based on a given numerical condition, values outside of that bound are considered irrelevant.

B.7 Sample Generation with Permutations

The input of this stage are the generated queries and the *concept/unit index*. For each quantity-centric query, a list of positive and negative samples is created by applying the numerical condition on the list of sentences from the index. The original positive and negative samples are then chosen at random from such a list. The same list is utilized as seed samples for data augmentation. Unit and value permutation are employed to generate augmented positive and hard negative samples. Hard negatives are created from positive samples, by permuating the unit or value to violet the query condition.

The steps are presented in Algorithm 2. Each sampling mechanism is encapsulated within a distinct function, and the final training samples are the union of all generation mechanisms. In the algorithm, v refers to a value, $vals$ to the list of the values of a given concept and unit, u to a unit, c to a condition, cn to a concept, and n to the sample size.

Algorithm 2 Sample Generation

```
function ORIGINAL_SAMPLING(s+, s−, n)
    return sample( s+,n), sample( s−,n)
end function

function UNIT_PERMUTATION(s+, n, u)
    su+ ← replace_same_unit_surface(s+, u)
    su− ← replace_other_unit_surface(s+, u)
    return sample( su+,n), sample( su−,n)
end function

function V_PERMUTATION(s+, s−, n, vals, c)
    su+ ← replace_with_positive_value(s−, v)
    su− ← replace_with_negative_value(s+, v, c)
    return sample( sv+,n), sample( sv−,n)
end function

conc_unit_dict ← concept/unit index
queries ← list of queries
n ← number of samples
for (cn, u, c, v) in queries do:
    s, vals ← conc_unit_dict[(cn, u)]
    s+, s− ← filter_based_on_condition(s, c, v)
    so+, so− ← ORIGINAL_SAMPLING(s+, s−, n)
    su+, su− ← UNIT_PERMUTATION(s+, n, u)
    sv+, sv− ← V_PERMUTATION(s+, s−, n, vals, v, c)
    sf+ = so+ ∪ su+ ∪ sv+
    sf− = so− ∪ su− ∪ sv−
end for
```

B.8 Sampling within Distribution

It is crucial that the permutation values obey the original value distribution of the corpus. The properties of concepts are often limited to a specific range, e.g., the value “10000” is an unreasonable unemployment rate. Moreover, certain values are on a discrete scale with limited options, e.g., “RAM of a laptop” is limited to distinct values such as 4,8, and 16. Assigning a random number outside this range, like 10, would be unrealistic. Therefore, for the synthetic data to obey the rules of the real-world datasets and reflect the distribution of different properties, the permuted values are chosen from the values observed in the corpus.

B.9 Down-sampling

If the number of available sentences in the positive and negative lists is smaller than the sample size, a *downsampling* procedure is implemented. When $|s_+| < n$ or $|s_-| < n$, we reduce the sample size to the smallest number of available samples.

C Evaluation

In this section, we present additional evaluations and implementation details. To reproduce the results or to access the trained model checkpoints and datasets, we encourage the reader to refer to our repository under `https://github.com/satya77/QuantityAwareRankers`.

C.1 FinQuant and MedQuant Datasets

In this section, we give an overview of the creation of the FinQuant and MedQuant evaluation benchmarks. FinQuant is created from a set of news articles in the categories of economics, science, sports, and technology, collected between 2018 and 2022. MedQuant contains TREC Medical Records (Voorhees, 2013) on clinical trials. Both datasets were split into sentences and processed to eliminate boilerplate HTML and headers. All sentences containing quantities were incorporated into the collection. The entire test data is manually created and tagged. In the following, we describe the query formulation and annotation task.

Query formulation: Given access to the compelete *concept/unit index* and the value distributions, annotators were tasked to formulate quantity-centric queries. During formulation, they were instructed to scan the entire index for possible synonyms for a given concept and keep track of the synonyms in a list. For example, if one chooses “Microsoft Surface Earbuds” with the unit “pound sterling”, the annotator scans the other concepts inside the *concept/unit index* that co-occur with “pound sterling” to detect synonyms or synsets, e.g., “Earbuds” and “Microsoft headphones” are related to the concept “Microsoft Surface Earbuds”. In the subsequent stage, the value distributions of all selected concepts are consolidated into one and presented to the annotator. The annotator is then instructed to choose three values for *equal*, *less than*, and *greater than* queries, in such a way that supporting sentences for the query are present within the value distribution. In the final stage, the annotator will formulate the query in natural text, e.g., “Microsoft Surface Earbuds lower than 179 pound sterling”. The annotators have access to the dictionary of surface forms for units and conditions to help with the query formulation.

Candidate list generation: For each query, a list of relevant sentences as candidates was generated using the *concept/unit index*. All sentences related to the concept and its synonyms were filtered based on the query value and condition. The filtering is done automatically based on the query value and numerical condition to lower the effort of annotation. We recognize that the quality of the candidate set relies directly on how effectively the quantity extractor captures associations between quantity and concepts. We observed that although the extractions in financial domain are of high quality, in the medical domain, several quantities were overlooked. In both datasets, there is no guarantee that the candidate list is comprehensive and covers all relevant instances.

Annotation: An annotation guideline was devised for consistent annotation of ambiguous cases and is published with the dataset. Annotators were presented with a list of candidate sentences for each query and were tasked to mark the relevant sentences. The marked sentences are used as ground truth for subsequent evaluation.

C.2 Semantic and Lexical Queries

The queries from the test set are categorized into four types: *seen*, *unseen*, *expansion*, and *w/o surface form*. The lexical queries fall under the categories of *seen* and *unseen*. For such cases, during query formulation, the annotators picked concepts from the *concept/unit index* without any change in their surface form. The concepts from the *unseen* category were removed from the index for data generation and training of the joint neural models. Therefore, this subset contains lexical queries that were not seen during training. For example, "YouTube channel" is a concept in the *unseen* subset, which means all instances of "YouTube channel" were removed from the *concept/unit index* before data generation.
Semantic queries contain the two subsets of *expansion* and *w/o surface form* and were slightly harder to formulate, thereby, fewer instances of them are present in the data. For *expansion* queries, a concept from the lexical set was chosen to expand to one of its supersets or synonyms. For example, "social media channel" is a semantic concept from "YouTube channel". These expansions were used to formulate queries that did not have a lexical match in the database and often included a superset of many concepts. In the case of "social media channels", the model should be able to retrieve other social media channels like "Facebook" as well as "YouTube". In the case of lexical models based on BM25, the difference is evident in Fig. 3a, where the models show great performance on the *seen* and *unseen* subset but if the same queries are converted to their semantic counterpart, as in *expansion*, the models fail to retrieve the correct result. *W/o surface form* are other semantic queries that were formulated independent of the lexical queries.

C.3 Implementation

The code is implemented in Python 3.10.9 and PyTorch 1.13.1. Sentence splitting and text cleaning were performed with SpaCy 3.6 [4]. As mentioned before we use the CQE library [5] for quantity extraction. Evaluation and metrics were computed with the help of `pytrec_eval` library (Van Gysel and de Rijke, 2018). In the following, we discuss the implementation details for each model separately.

BM25 models: We use the Okapi BM25 package [6] for all BM25 variants. The QBM25 and BM25_{filter} are variations of Okapi BM25 designed to include a numerical index for ranking and filtering. The parameters of BM25 were tuned to each of our datasets separately, as presented in Table 4. The latency values are computed with plug-ins for an Opensearch [7] instance on a desktop computer with 16GB of RAM. In comparison to the dense models, the lexical models do not require specific hardware architectures.

Table 4: Hyper parameters of BM25-based models on the benchmark datasets.

	FinQuant	MedQuant
BM25	b = 0.5, k1 = 0.5,	b = 0.5, k1 = 0.5
BM25_{filter}	b = 0.75, k1 = 1.5	b = 0.75, k1 = 1.5
QBM25	b = 0.5, k1 = 0.5	b = 0.5, k1 = 0.75

Cohere baseline: We used the Cohere API [8] for Cohere_{v3} embeddings. Query embeddings were used to encode the queries and document embeddings to encode the collection.

ColBERT models: (Khattab and Zaharia, 2020) supplied the trained checkpoint for the base

[4] `https://spacy.io/` DLA: 27.05.2024
[5] `https://github.com/vivkaz/CQE` DLA: 27.05.2024
[6] `https://pypi.org/project/rank-bm25/` DLA: 27.05.2024
[7] `https://opensearch.org/` DLA: 27.05.2024
[8] `https://cohere.com/` DLA: 27.05.2024

ColBERT model. For fine-tuning augmented data, the model was initialized with this base checkpoint. The checkpoint was employed for the evaluation of both ColBERT and QColBERT. ColBERT$_{ft}$ was fine-tunned using the training script from the official repository [9]. The code in the repository was modified to establish an endpoint for QColBERT, incorporating a quantity index. We did not perform extensive hyperparameter tuning except for the learning rate and used the parameters advised by the authors for both FinQuant and MedQuant datasets. We fine-tuned the joint ColBERT$_{ft}$ for 2 epochs, with a batch size of 256 and a learning rate of 1e-05 on a server with four A-100 GPUs and 40GB of memory. The evaluation and benchmarking for latency were performed on the same server, utilizing all four GPUs.

SPLADE models: SPLADE$_{ft}$ was also fine-tuned using the training script by the authors [10]. The pre-trained checkpoint was acquired from HuggingFace [11] and utilized for both the SPLADE model and QSPLADE. Scripts from the official repository were adjusted to add a quantity index for QSPLADE. Similar to ColBERT, we conducted limited hyperparameter tuning, mainly focusing on the learning rate. We fine-tuned SPLADE$_{ft}$ for 2 epochs using a batch size of 240, a learning rate of 2e-5, and a weight decay of 0.01. The fine-tuning was conducted on a server with four A-100 GPUs and 40GB of memory. The evaluation and benchmarking for latency were performed on the same server, utilizing all four GPUs.

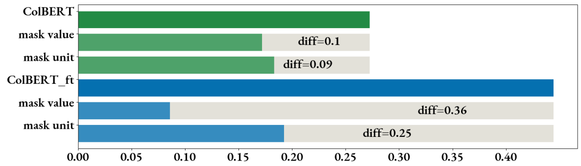
For all disjoint rankers, QBM25, QColBERT, and QSPLADE, the quantity impact parameter of α is set to 1, such that the impact of term and quantity ranking are equal.

Generated data: Based on the combination of augmentation methods the size of training data would vary. In all cases, we saved a small sample of 1,000 queries for validation. There were 40,732 and 20,376 concept and unit pairs considered for query generation in FinQuant and MedQuant, respectively. If concept expansion is applied these numbers would double to account for queries on expanded concepts. We set the sample size n to 2, meaning that for each query two positive and the negative samples were chosen from the data without augmentation. As a result, based on augmentation methods, additional $n = 2$ samples would be added for unit and value permutation, a total of $3n$ per query.

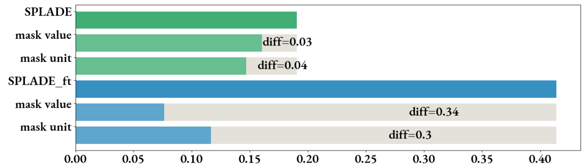
[9] https://github.com/stanford-futuredata/ColBERT DLA:27.05.2024

[10] https://github.com/naver/splade DLA:27.05.2024

[11] https://huggingface.co/naver/splade-cocondenser-ensembledistil DLA:27.05.2024



(a) ColBERT based models



(b) SPLADE based models

Figure 8: The effect of task-specific fine-tuning on models attention to quantity tokens. In the masked variants either the unit or the value of the sentences in the collection is masked.

C.4 Effect of Fine-tuning

To assess the impact of task-specific fine tuning on the internal ranking strategy of the dense models, we evaluate two masked versions of the data.
Mask value: In this scenario, we mask all values in the collection with the `[MASK]` token before running the evaluation. This task aims to determine the extent to which the model depends on the value token for retrieving the correct sentence.
Mask unit: Here, we mask unit tokens in the collection before running the evaluation with `[MASK]` token. This task is intended to observe the impact of unit comparison on the final ranking.
We compare the base version of the dense models with their fine-tuned version on the different masking of the FinQuant dataset. The results for the ColBERT models are shown in Fig. 8a and for SPLADE models in Fig. 8b. In both cases, the fine-tuned version exhibits a more significant drop in performance compared to the base models when quantity tokens are masked. This indicates that after fine-tuning, the model becomes more dependent on the quantity tokens, namely, values and units, in the text to identify the relevant sentence.