Numbers Matter! Bringing Quantity-awareness to Retrieval Systems

Anonymous ACL submission

Abstract

Quantitative information plays a crucial role in understanding and interpreting 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 data are available under https://github.com/filled_in_later.

1 Introduction

Despite advances in semantic search, and sophisticated neural network architectures, handling quantitative information in text remains challenging. Specifically with 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 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 for specific tasks and not semantic search in general. This applies to neural models supporting Information Retrieval (IR), which are trained on general-purpose data without the focus on 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 (Spokoyny et al., 2022; Jin et al., 2021), 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, 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 an unsupervised and heuristic model utilizing an index structure, compatible with various lexical and semantic IR systems. Due to the independence assumption, the connection between quantities and surrounding text is somewhat lost. Therefore, for joint ranking, we aim to learn quantity-aware document and query representations through task-specific fine-tuning of neural IR models. Additionally, we introduce two novel benchmark datasets for quantity-centric ranking, specifically focusing on queries involving numerical conditions, in the domains of finance and medicine. We evaluate the performance of our systems against 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) focuses 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 their approach 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, (Rybinski et al., 2023) and (Li et al., 2021) perform numerical summarization on unstructured text in 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 (Maiya et al., 2015; Fontoura et al., 2007; Agrawal and Srikant, 2003). 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) is 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 (Spokoyny et al., 2022; Jin et al., 2021; Thawani et al., 2021; Sundararaman et al., 2020; Jiang et al., 2020; Nogueira et al., 2021; Spithourakis and Riedel, 2018). These models 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 architecture change or training a 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 don't 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 (Ho et al., 2019; Almasian et al., 2022), 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 the query value. The relevance, $r(s_i|X)$, of sentence s_i to query X is denoted 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 document to their representations. Here, we explore different ways to define τ, which can be an embedding vector or a heuristic scoring function.

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

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. We point out the limitation of the disjoint ranking and propose a quantity-centric fine-tuning paradigm for neural IR systems for the joint ranking of quantity and textual content.

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, and 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). However, this module can be substituted with any alternative 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 and sentence values under the query condition. Then, the relevance of a sentence can be the summation of (1) textual similarity, and (2) proximity of quantities in a sentence and a query, denoted in Eq 2. Note that here, sim computes the similarity of search terms to 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 necessary created from the same model. If the query is not quantity-centric, simply by removing the quantity score sim_c, the models fall back to term scoring.

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

In the following, we describe the computation of term (1) , and quantity (2) scorings, where (1) $sim(\tau(T_x), \tau(T_i))$ and (2) $sim_c(\tau(q_x), \tau(Q_i)$. The general pipeline is depicted in App A.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. CQE 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 is built from this output and resembles a lexical index. Here, 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, resulting in different scores for the same values under different numerical 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 $qs(s, c, X)$ as simply qs.

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

Φ_c consists of 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.2). Simply by changing the Φs, results can be rearranged, independent of training data that might introduce bias for a specific sorting preference. 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. The Φs are defined in Eq. 4. v_x is the query value and v_i is the sentence value.

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

$\Phi_{=}$ 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.

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 set to zero. The $\text{BM25}(s_i, T_x)$ 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 term scoring. This choice is due to the same model being used for joint quantity ranking, where token-level interactions are crucial. Contextualized term score is computed with the similarity computation between token embeddings of query and sentence, as in Eq 6. ColBERT utilizes two BERT (Devlin et al., 2019) encoders for query and document (sentence), where each encoder outputs a list of token embeddings.

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

The term score comes from the MaxSim operation between the query and sentence embeddings. MaxSim calculates an unbounded score for the maximum cosine similarity among the token embeddings. To normalize this score, we require 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 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, α controls the impact of quantity scoring.

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

Note that the 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. Instead of term frequencies inside the index, 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 inside this vector are the term importance. Then the relevance of the query to a sentence is based on the sparse dot product of this vector, as denoted 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, 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 the relevance of quantities and terms can 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 and 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. To investigate this, we propose a data generation approach to address following problems.

First is the inability to perform value comparisons given numerical conditions. E.g., in the example above, the models ignore the *less than*, condition and focus on the semantic similarity of query text and sentence. Second, the semantic similarity of units is not well-defined. In the example above, results with "dollar" and other currencies receive high scores due to the context similarity of the units. Refer to App B.6 for a 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 used in the disjoint setting, ColBERT, and SPLADE, on synthetic data for quantity-centric IR [1]. The data generation pipeline has three stages 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 structures 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 given in an algorithm in the App B.2. In the following, 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 "euro". `unit_before` and `unit_after` account for symbols appearing before, e.g., "€" and abbreviations after a value, e.g., "EUR", respectively.

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 of *concept/unit index*:

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

Less and greater than queries: For these bounds, optimal candidates are close to the average of the value distribution, such that when the numerical condition is applied more sentence 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 on value selection.

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

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 *greater than* condition. For this purpose, 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

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

5 dollar/month". Relying exclusively on keywords in sentences poses a 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, namely 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 for each numerical condition.

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. An overview of the sample generation pipeline and an algorithmic view is presented in App B.7. In the following, we describe each step in detail.

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

Positive and Negative Sentences List: The obtained sentences are divided into positive s_+ and negative s_- lists, based on the numerical condition. s_+ contains sentences, where the values in them satisfying the condition and s_- contains sentence 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 information on the sample size.

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, while negative samples include surface forms of units in the same family as the query unit, creating negatives.

- A positive sample, s_{u+}, is formulated by substituting the unit in a positive sentences, s_+, with other surface forms of unit in query u_i.
- A negative sample, s_{u-} is created by replacing the unit in a positive sentences, s_+, with a surface form of a unit different from query unit ,u_i, but belonging to the same family. The unit families are grouping 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 formulated 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 invalidates the quantity condition.

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

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 $s_{f-} = s_{o-} \cup s_{u-} \cup s_{v-}$.

The models reported in the evaluation use a combination of original sampling with unit permutation and concept expansion on the query. Value permutation did not show stable performance gains, which we attribute to the difficulty of numerical representations in dense models. For more discussion on this matter and ablation study of augmentation methods refer to App C.5.

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 the numerical indices from databases, focusing on filtering rather than ranking.

Table 1: Query types in FinQuant and MedQuant.

	FinQuant	MedQuant
Total queries	420	210
Sentence in corpus	306,291	153,252
Per condition	140	70
Keyword-based queries	300	120
Semantic queries	120	90

Neural models include the trained checkpoints of SPLADE and ColBERT as well as Cohere$_{v3}$ [2]. Cohere$_{v3}$ embeddings are included to show that even industry-level models trained on extensive data still lack quantity-centric understanding.

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, refer to App C.1. The data is annotated by the 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. 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 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, impacting performance comparisons later on. However, we find it important to report results on both datasets, making the reader aware of the lesser 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 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, the re-ranking with the quantity score proves beneficial.

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

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 minimal performance drop, suggesting that the models learn patterns for quantity-centric queries without mem-

[2] https://cohere.com/embeddings DLA: 10.02.24

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

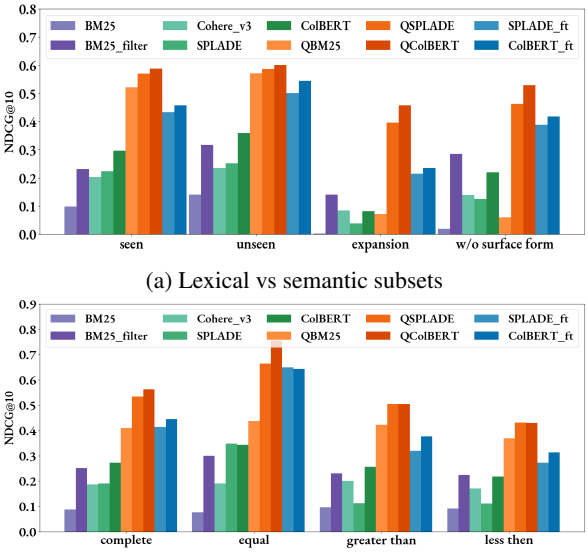
	Model	latency (ms)	FinQuant				MedQuant			
			P@10	MRR@10	NDCG@10	R@100	P@10	MRR@10	NDCG@10	R@100
baselines	BM25	9	0.06	0.14	0.09	0.47	0.04	0.11	0.07	0.37
	$BM25_{filter}$	9	0.14	0.32	0.25	0.60	0.08	0.19	0.15	0.48
	$Cohere_{v3}$	-	0.14	0.22	0.19	0.27	0.10	0.17	0.15	0.25
	SPLADE	26	0.10	0.24	0.19	0.53	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
joint	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**†
disjoint	$SPLADE_{ft}$	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

orizing common queries.

4.4 Lexical vs Semantic Queries

Fig 1a shows NDCG@10 of all models on lexical and semantic subsets of the FinQuant. The *seen* and *unseen* are lexical queries and *expansion* and *w/o surface form* represent semantic queries. For the details of their distinction refer to App C.2. Interestingly, the disjoint ranking using dense models captures both semantic similarity and quantity understanding. QBM25 performs equally well as dense models in lexical queries but significantly worse on semantic ones. Joint rankers outperform base models in both lexical and semantic queries but lag behind disjoint models.

Fig 1b 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.



(a) Lexical vs semantic subsets

(b) Subset with different numerical conditions

Figure 1: Performance on different subsets of FinQuant.

5 Conclusion and Ongoing Work

This work introduces two methods to integrate quantity understanding into existing retrieval systems. The disjoint approach is an unsupervised and heuristic method, while the joint approach involves fine-tuning on quantity-centric synthetic data to enhance quantity understanding. The disjoint scoring can be combined with any lexical or semantic matchers without the change in their architecture or need for fine-tuning, showing consistently good performance regardless of data distribution. Moreover, the notation of quantity proximity is easily altered by changing the quantity scoring function, leading to great flexibility in terms of different sorting of results. However, the quantity index introduces an overhead in query latency, is sensitive to errors from quantity extraction and the independence assumption leads to possible concept-quantity mismatch in the results. Conversely, the joint models are better at finding concept and quantity associations but their overall performance is lower. Yet, if one does not want to be dependent on an external index and quantity extractor, the fine-tuning on synthetic data can enhance quantity awareness to some extend. We also introduce two benchmark datasets and evaluated our methods against multiple baselines. In future, we explore the impact of numerical embeddings in 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 about 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 the query 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".

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 don on CPU. Moreover, 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 is not detected by the extractor it 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. In this work, we do not discuss models that deal with *ranges* and *negations*. Adding 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.

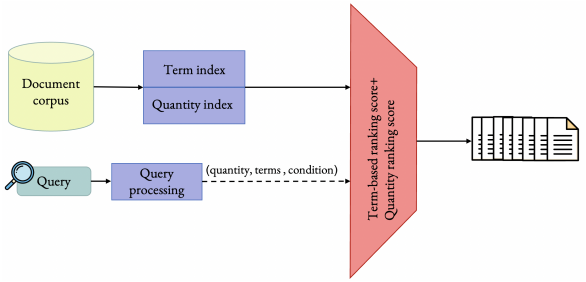


Figure 2: General pipeline of the disjoint quantity-ranking approach, where a separate quantity index is responsible for computing quantity proximity and a term-based lexical or semantic index is used to compute the similarity of the search terms to sentences.

A Disjoint Quantity Ranking

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

A.1 Disjoint Quantity Ranking Pipeline

The general pipeline for the disjoint quantity-ranking model is shown in Figure 2. The query is processed into quantity, search terms, and conditions, using CQE or similar packages. The document corpus is indexed separately for terms and quantities, whereas the term-based index can be a traditional lexical index or a vector database. This term-based index retrieves semantically or lexically similar sentences. From the retrieved sentences, the quantity index identifies values that share the same unit as the query, computing proximity based on the provided condition. The final ranking combines scores from term-based and quantity ranking.

A.2 Optimal Sorting

Although all the sentences that satisfy a numerical condition and have the correct concept and unit are potentially relevant, 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. Disjoint approach is flexible concerning different sorting. By switching a scoring function, the results can be rearranged. Joint model 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 the additional material related to the joint quantity ranking model.

B.1 Concept-unit Index

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

```
{("cannabis company","cent per share"):
{"values":[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 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., value "17.0" is repeated twice as it occurs in two distinct sentences. The creation steps are depicted in Figure 3. The corpus is analyzed with CQE to extract values, units, and concepts from each sentence, where sentences sharing the same unit and concept are grouped into a list, along with values represented as a distribution.

B.2 Query Generation

The complete query generation pipeline is depicted in Figure 4. The *concept/unit index* is used to select values and units for numerical conditions. Additionally, a large language model is used to expand concepts for semantic queries. The template generation block combines all the outputs to formulate three queries for each unit and concept pair. To generate expanded concepts, a new query value is chosen from the value distribution, leading to the formulation of a new set of queries. Additionally, 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 the value, u to the unit, c to the condition, and cn to the concept.

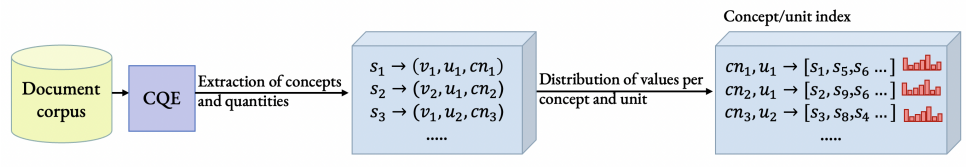


Figure 3: Overview of the quantity tagging step and creation of *concept/unit index* structure.

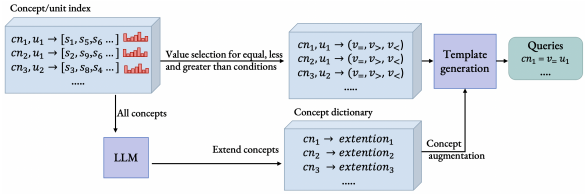


Figure 4: Overview of the query generation pipeline, using *concept/unit* index and a large language model for concept expansion.

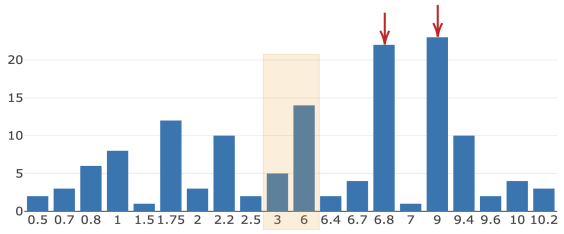


Figure 5: An example of choosing query values for *equal* and bound-based conditions.

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

B.3 Choosing the Right Query Value

Each entry in the *concept/unit index* points to the sentences and list of values in those sentences. For the data augmentation to work, 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 Figure 5. For the *equal* query, the challenge is to find enough positive samples, since there is an abundance of not equal values in each distribution. In Figure 5, 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 a 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 avoid too few positive or negative samples.

B.4 Dictionary of Numerical Conditions

A non-comprehensive dictionary of surface forms for numerical conditions is shown 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

[3] https://openai.com/ DLA:11.02.2024

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 place-holder concept is replaced with a concept from the *concept/unit index*.

For finance 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 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} =
```

B.6 When Semantic Search Back-fires

Semantic retrieval systems consider an entire context to find a fuzzy 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 compose 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 which makes the car accelerate. Another common problem is with currencies. Given that monetary values often appear in similar contexts, it becomes challenging for the 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.

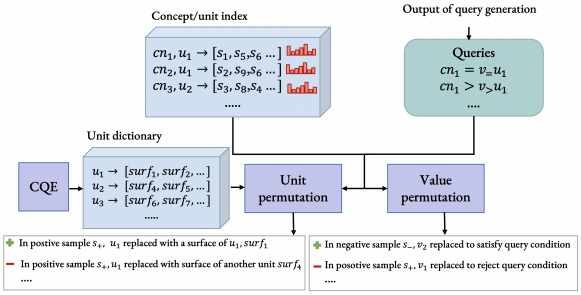


Figure 6: Overview of the sample generation using value and unit permutation.

B.7 Sample Generation with Permutations

An outline of sample generation pipeline is shown in Figure 6. The input of this stage is the generated queries and the *concept/unit index*. For each quantity-centric query, a list of positive and negative samples are 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 negative are positive samples, where the unit or value is perturb 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 value, $vals$ to list of value of a given concept and unit, u to unit, c to condition, cn to concept and n to 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)
  $s_{u+} \leftarrow$ replace_same_unit_surface(s_+, u)
  $s_{u-} \leftarrow$ replace_other_unit_surface(s_+, u)
  return sample(s_{u+},n), sample(s_{u-},n)
end function

function V_PERMUATION($s_+, s_-, n, vals, c$)
  $s_{u+} \leftarrow$ replace_with_positive_value(s_-, v)
  $s_{u-} \leftarrow$ replace_with_negative_value(s_+, v, c)
  return sample(s_{v+},n), sample(s_{v-},n)
end function

conc_unit_dict $\leftarrow$ concept/unit index
queries $\leftarrow$ list of queries
n $\leftarrow$ number of samples
for (cn, u, c, v) in queries **do**:
  $s, vals \leftarrow$ conc_unit_dict[(cn, u)]
  $s_+, s_- \leftarrow$ filter_based_on_condition(s, c, v)
  $s_{o+}, s_{o-} \leftarrow$ **ORIGINAL_SAMPLING**(s_+, s_-, n)
  $s_{u+}, s_{u-} \leftarrow$ **UNIT_PERMUTATION**(s_+, n, u)
  $s_{v+}, s_{v-} \leftarrow$ **V_PERMUTATION**$(s_+, s_-, n, vals, v, c)$
  $s_{f+} = s_{o+} \cup s_{u+} \cup s_{v+}$
  $s_{f-} = s_{o-} \cup s_{u-} \cup s_{v-}$
end for

B.8 Sampling within Distribution

It is crucial that the permuted 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 unreasonable for percentage rate of unemployment. 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 rule of the real-world dataset and reflect the distribution of different properties, the permuted value 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, access model checkpoints and datasets, we encourage readers to refer to our repository.

C.1 FinQuant and MedQuant Datasets

In this section, we give an overview of the creation the FinQuant and MedQuant evaluation benchmarks. FinQuant is created from a set of news articles in categories of economics, science, sports, and technology, collected between 2018 and 2022. MedQuant contains TREC Medical Records (Voorhees, 2013) on clinical Trails. Both datasets were split into sentences and processed to eliminate boilerplate HTML or headers. All sentences containing a single quantity 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 *concept/unit index* and the value distributions, annotators were tasked to formulate quantity-centric queries. They were instructed to scan the entire index for possible synonyms to a given concept when formulating a query 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* with "pound sterling" as unit to detect relevant ones, e.g., "Earbuds", "Microsoft headphones". 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 is 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 query formulation.

Candiate list generation : For each query, a list of candidates relevant sentences 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 for financial data were 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 a 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 at ground truth for subsequent evaluation.

C.2 Semantic and Lexical Queries

The queries 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 the change in surface form. The concepts from the *unseen* category, were removed from the index for data generation and training of the joint neural models. Therefore, it 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 fall under the categories 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 Figure1a, where the models show great performance on *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 PyTroch 1.13.1. The general 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 15GB 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 the document embeddings to encode the collection.

ColBERT models: (Khattab and Zaharia, 2020) supplied the trained checkpoint for the base ColBERT model. For fine-tuning on augmented data, the model was initialized with this base checkpoint. The checkpoint was employed for

[4] https://spacy.io/ DLA: 11.02.2024
[5] https://github.com/vivkaz/CQE DLA: 11.02.2024
[6] https://pypi.org/project/rank-bm25/ DLA: 11.02.2024
[7] https://opensearch.org/ DLA: 11.02.2024
[8] https://cohere.com/ DLA: 11.02.2024

the evaluation of both ColBERT and QColBERT. ColBERT$_{ft}$ was fine-tunned using the training script the 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 mainly 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 Hugging Face [11] and utilized for both the SPLADE model and QSPLADE. Scripts from the official repository were adjusted to add numerical 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.

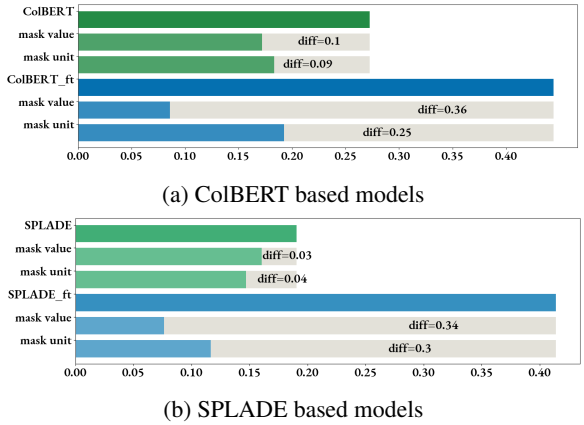
For all disjoint rankers, QBM25, QColBERT, and QSPLADE the quantity impact parameter of α is set to 1, such that the impact of term and quantity ranking are equal.

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 1000 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 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 $3n$ per query.

[9] https://github.com/stanford-futuredata/ColBERT DLA:11.02.2024

[10] https://github.com/naver/splade DLA:11.02.2024

[11] https://huggingface.co/naver/splade-cocondenser-ensembledistil DLA:11.02.2024



(a) ColBERT based models

(b) SPLADE based models

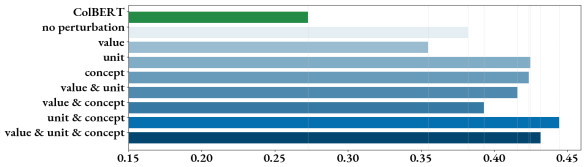
Figure 7: The effect of task-specific fine tuning on models attention to quantity tokens. In the masked variants either the unit or the value of the sentences in the collection is masked.

C.4 Effect of Fine-tuning

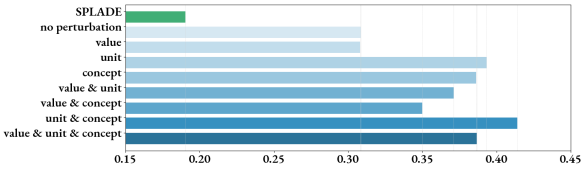
To assess the impact of task-specific fine-tuning on the internal ranking strategy of the dense models, we evaluate two masked versions of the data.
Mask value: In this scenario, we mask all values in the collection with the `[MASK]` token before running the evaluation. This task aims to determine the 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 Figure 7a and for SPLADE models in Figure 7b. In both cases, the fine-tuned version exhibits a more significant drop in performance compared to the base models. This indicates that after fine-tuning, the model becomes more dependent on the quantity of tokens, values, and units, in the text to identify the relevant sentence.

C.5 Ablation Study on Augmentation Methods

To check the effect of different augmentation strategies, we perform an ablation study, by fine-tuning



(a) ColBERT$_{ft}$



(b) SPLADE$_{ft}$

Figure 8: Ablation study on different augmentation methods, where *value* and *unit*, refer to value and unit permutation and *concept* refers to concept expansion.

the neural models on data generated using a combination of different strategies. The main points of variabilities are concept expansion in the query generation process and value and unit permutation for sample generation. The results for ColBERT$_{ft}$ and SPLADE$_{ft}$ on FinQuant dataset is demonstrated in Figures 8a and 8b, respectively. *no perturbation* refers to the case where no data augmentation was applied and only the positive and negative samples from the original sampling are used for training.

An interesting trend is the detrimental effect of value permutation. The value permutation on its own enhances the performance of the base model. However, as soon as it is accompanied by other augmentation methods the performance degrades slightly. The best combination for both the SPLADE and ColBERT model is unit permutation and concept expansion, both of these augmentation techniques on their own also provide a larger boost in comparison to value permutation. To this end, the variant of the models presented for evaluation as ColBERT$_{ft}$ and SPLADE$_{ft}$ are trained on unit permutation and concept expansion subset. We find this behaviour rather surprising and counter-intuitive. Usually, the performance of neural models increases with the amount of data presented for a given task, however, perturbing the values does not seem to enhance the performance as expected. This can be related to the internal representation of the neural models, which is hindering their ability to correctly learn quantity semantics. In future work, we aim to test the effect of dedicated numerical embedding and language models for this task.