Applying Structural and Dense Semantic Matching for the ARQMath Lab 2022, CLEF

Wei Zhong[1,2], Yuqing Xie[1,2] and Jimmy Lin[2]

[1](two authors contributed equally to this work)
[2]David R. Cheriton School of Computer Science, University of Waterloo

Abstract

This work describes the participation of our team in the ARQMath 2022 Lab, where we have applied two highly complementary methods for effective math answer and formula retrieval. More specifically, a lexical sparse retriever (Approach Zero) capable of first-stage structure matching is combined with a fine-tuned bi-encoder dense retriever (ColBERT) to capture contextual similarity and semantic matching. The dense retrieval model is further pretrained to adapt to math domain content containing LaTeX tokens. In the Open Domain QA task, we take an extractive approach and filter sentences using heuristic rules applied to top-ranked answers returned from our retrievers. We provide an analysis of both the effectiveness and efficiency of our models. In this contest, our effectiveness is ranked at the top among all three tasks.

Keywords

Mathematics information retrieval, structure search, dense retrieval, math-aware search.

1. Introduction

The steady growth of scientific publications and the need to retrieve math-related content by formulas have attracted the attention of many researchers into the Mathematics Information Retrieval (MIR) field recently. The core task in MIR is to retrieve relevant information from documents that contain math formulas. However, the heterogeneous data presented in math content (including rich-structured formulas and their textual context) require special treatment to create a truly effective search engine. The difficulty arises not only because the usage of math notation demands a special tokenizer for processing, but also because of the rich structures and special semantic properties implied by math languages such as expression commutativity and symbol substitution equivalence. Furthermore, the general challenge in understanding math content in MIR imposes another hurdle compared to other IR tasks.

The ARQMath Labs have been one of the few tasks facing this challenge. The ARQMath-1 (2020) [1] and ARQMath-2 (2021) [2] include two tasks: Task 1 is a Community Question and Answer (CQA) task that asks to retrieve relevant answer posts from a limited set of Math StackExchange (MSE)[1] corpus between the year 2010 to 2018, given queries of real-world questions sampled from later-year MSE threads. Task 2 is a formula-centered task where it asks to return relevant formulas (considering their context) in the documents given one specified

CLEF 2022: Conference and Labs of the Evaluation Forum, September 5–8, 2022, Bologna, Italy
w32zhong@uwaterloo.ca (W. Zhong); yuqing.xie@uwaterloo.ca (Y. Xie); jimmylin@uwaterloo.ca (J. Lin)

[1]https://math.stackexchange.com/

query formula appeared in a Task 1 topic. To encourage formula diversity, this task requires at most 5 visually distinct formulas to be returned, otherwise, the result will not be judged. This time, i.e., ARQMath-3, an additional Open Domain QA task (Task 3) has been introduced. Task 3 requires participants to return a single answer for each topic in Task 1, and the answer can be automatically generated or extracted from existing data, potentially outside the ARQMath collection.

To deal with formulas in math content, not only do the structured expressions need to be paid special attention to but also the context where a math formula occurs needs to be considered for a better understanding of the formula itself. Moreover, the context helps to discover math formula similarities even if formulas look different in structure. This resembles the synonym issue in regular full-text retrieval, where exact lexical matching prevents the return of semantically relevant documents having only synonyms to the query keywords. In both cases, recent studies using bi-encoder dense retrieval models [3, 4, 5, 6, 7, 8] have shown success for in-domain effectiveness and the ability to discover semantic similarities even if content lacks of lexical agreement. On the other hand, the structure of math formulas itself is an important aspect of similarity in MIR. For example, being a large substructure of another math expression is a good indicator of similarity.

In this work, we combine a structure-aware search engine with a bi-encoder dense retriever (See Lin [9] for this classification) to capture both the important structure similarity and semantic similarity. According to our recent findings [10], we adopt the ColBERT model [4] due to its high effectiveness demonstrated in our evaluation of previous MIR tasks.

2. Related Work

Early work on MIR simply applies specialized tokenizers to handle math formulas [11]. Later, different intermediate tree representations are utilized to extract features for capturing structure similarities.

The Operator Trees (OPT) representation represents a math formula by identifying operators and operands in the expression, and constructing a tree recursively where each internal node is the operator of its children's operands. To our best knowledge, Hijikata et al. [12], Yokoi and Aizawa [13] are the first to use the OPT representation for math retrieval. They extract leaf-root paths from OPT (alternatively in representational MathML) as features that are invariant to operand positional mutation (e.g., due to commutativity) for retrieval. Later, Zhong et al. [14, 15] extend OPT leaf-path matching to more strict structure matching with real-time efficiency. Their system (i.e., Approach Zero [2]) defines a meaningful metric for formula structure similarity by finding the maximum common subtree(s) between two math formula OPTs. This approach has achieved the best effectiveness for the formula search task in the ARQMath 2021 [2].

On the other hand, the Symbol Layout Tree (SLT) [16, 17] represents a lower level structure semantics for math formulas. Similar to the LaTeX representation, it only captures the layout or the topology of a formula. This creates an advantage of little ambiguity in parsing. SLT is adopted as the main representation by a line of MIR works, e.g., the Tangent and Tangent-S systems [18, 19, 20], and the Tangent-L or the MathDowsers system [21, 22, 23, 24, 25]. Local

[2] Approach Zero or `approach0` gets its name from the word "asymptotics", the core concept to define a limit.

features such as symbols on adjacent nodes or nodes within a distance window and their spatial relations are together tokenized into *math tuples*[3] and used for retrieval. As an example, the up-to-date MathDowsers system extracts more than five types of features from SLT [25].

There are other systems such as the MCAT system [26] and the Tangent-S system [27] which incorporate both SLT and OPT representations. Both use linear regression to interpolate scores contributed from SLT and OPT features. The Tangent-CFTED system [28], an upgraded version of Tangent-S, further applies the FastText algorithm [29] to learn structural embeddings from SLT and OPT local features for candidate retrieval, then it reranks them by tree edit distance.

More recently, Transformer models for MIR have entered the MIR domain. Although it has been shown Transformer-based language model may still be relatively weak at math tasks [30, 31], these Transformer models have nevertheless demonstrated their good effectiveness at MIR. The MathBERT model [32], evaluated on the NTCIR-12 collection [33], introduces structure mask pretraining on top of the pretraining objectives used in BERT [34, 35]. It uses the last two layers' feature vectors for reranking. The previous ARQMath tasks [1, 2], however, have witnessed more widespread use of deep models in MIR. Among them, Novotnỳ et al. [36, 37] use a SentenceBERT-based Transformer (i.e., CompuBERT) [38] to regress QA pair scores based on user-generated data in the original MSE thread. And Rohatgi et al. [39, 40] reranks search results using the full token embeddings generated from a pretrained RoBERTa Transformer. The DPRL QASim method [41] uses two Transformers as similarity assessors, one question-question SentenceBERT [38] assessor pretrained on the Quora website and fine-tuned using related/duplicate links on the MSE website, as well as a question-answer TinyBERT [42] assessor pretrained on the MS-MARCO dataset [43] and fine-tuned on the ARQMath-1 training data. The similarity produced by QASim is a product of these two assessor scores where the question-question model evaluates the topic question and the question to which the document answer is given. Similarly, the TU_DBS systems [44, 45] uses a cross encoder as a major model in the ARQMath 2021, moreover, they apply the deep model to the Task 2 of formula retrieval. For the backbone model, they use an ALBERT Transformer [46] further pretrained on the ARQMath corpus directly with a 512 maximum token input.

However, the aforementioned models are either mostly cross encoders that require a full pass through the Transformer model to evaluate a pair of query and document, or they only use pretrained model embeddings without finetuning for better QA similarity assessment. As a result, they have to either only evaluate partial collections (e.g., considering only answers shared at least one tag with the topic), or use a simplified model architecture (e.g., using the TinyBERT or ALBERT instead) for fast inference on the million-scale ARQMath dataset. Prior to this ARQMath-3 Lab, to our best knowledge, only the CompuBERT model [36, 37], the ColBERT model [4] explored by the TU_DBS system [45], and our recent work [10] have used fine-tuned bi-encoder Transformer models (which are practically efficient) in the MIR domain. In this work, we will incorporate the ColBERT model into our structure-aware search system Approach Zero [14, 15] for the ARQMath-3 Lab.

[3] See an open-source parser package for examples: https://github.com/fwtompa/mathtuples

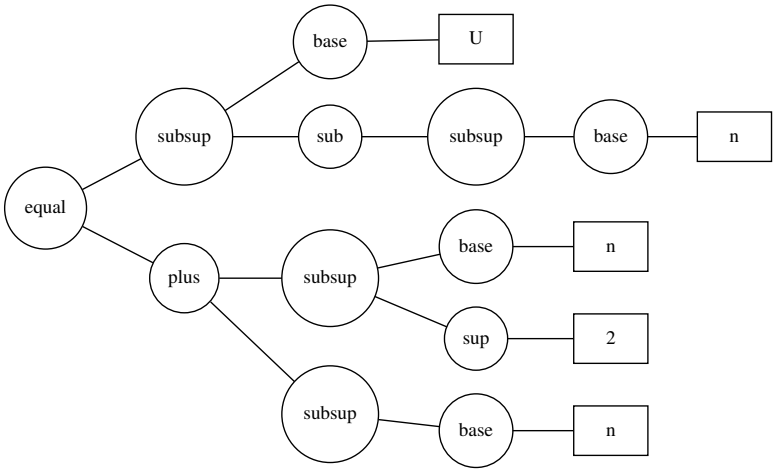

Figure 1: Operator Tree representation for formula "$U_n = n^2 + n$" (Topic B.285). Operator and leaf (i.e., operand) are denoted by circle and box respectively. In order to improve recall, operands with or without subscript (sub) or superscript (sup) are represented canonically under a subsup token.

3. Models

3.1. Lexical and Structural Search

Similar to our approach in the ARQMath 2021 [47], we use the Approach Zero system [14, 15] for searching for lexical text words and structural math formulas. Approach Zero constructs a customized OPT representation internally to extract structure features, specifically, it uses leaf-root path extracted from OPT and their path prefixes as "keywords". Figure 1 shows an example of OPT we use for representing an ARQMath topic formula. Compared to the representation we use in 2021 [47], it no longer requires a "sign" node on top of an operand, instead, we downplay its importance and include the operand sign into the "fingerprint" of the path where symbols are hashed into a single value to be scored and counted for symbol similarity rather than for structure similarity. Moreover, we have fixed a few inconsistencies in the grammar reduction rules which we use to construct the OPT tree. These engineering aspects of improvements, including a few bug fixes, turn out to greatly impact the effectiveness.

Given extracted leaf-root paths (and their prefixes), we tokenize all the nodes along each path to boost search recall. These paths are used as the vocabulary keys in a specialized inverted index [15] for mapping any query path (extracted from the OPTs of query formulas using the same way) to the corresponding posting lists which store relevant document path information. More specifically, each posting list item includes document ID, root-end node IDs, formula ID, formula length, the frequencies of paths under each subtree, leaf-end symbols (i.e., operand symbols), and path fingerprints. The matched query and document paths are evaluated one formula at a time at query processing.

For structure similarity, we calculate a common structure score $w(Q^{(m)}, D^{(n)})$ of node m in query formula Q and node n in document formula D by summing the number of common

paths under each substructure and weighted by a path idf [47]:

$$\mathrm{PathIDF}_p = \log \frac{N}{df_p} \tag{1}$$

where N is the total number of paths in the index and df_p is the document frequency of path p. The overall structure score $w^*(Q, D)$ is given by the maximum common structure score between query and document formulas:

$$\begin{aligned} w^*(Q, D) &= \max_{m,n} \; w(Q^{(m)}, D^{(n)}) \\ &= \max_{m,n} \sum_t \min(\left|Q_t^{(m)}\right|, \left|D_t^{(n)}\right|) \cdot \mathrm{PathIDF}_t \end{aligned} \tag{2}$$

where $\left|Q_t^{(m)}\right|$ and $\left|D_t^{(n)}\right|$ are the number of paths which have the same tokens (or vocabulary key) t under query node m and document node n respectively, and we look up them at retrieval time by grouping hit document paths by their root-end nodes and their (indexed) frequency of path t. The modeled structure similarity $w^*(Q, D)$ can also be viewed as the sum of path idf in the largest common subtree between query and document formula OPTs. This resembles the tf–idf scoring except for the "term frequency" here counts for common structure "width", i.e., the number of matched leaves in the common subtrees.

In addition to the aforementioned structure weight, we multiply a few other factors to the path idf for the final similarity score. First, paths in the maximum common structure are paired by their symbols (if they have the freedom to match another path of a different symbol in the counterpart). Then, the operand symbol and the fingerprint value associated with each path will determine the *symbol score* by summing the match points by the following rules:

- 1 point if both the operand symbol and the fingerprint match
- a lower point b_1 if only operand symbol match
- a nonzero base point b_2 otherwise ($b_2 < b_1$)

where the fingerprint is a hash value of the symbols of up to 4 operator nodes on top of the path leaf, it also takes into account the sign value (i.e., $1, -1$) induced for each operator (this includes the sign of the operand itself since the sign of an operand is induced into the `subsup` node which is always placed on top of an operand). A greedy matching algorithm `MarkAndCross` [48] is used to pair them such that a higher number of points is likely achieved, then we normalize it with the number of matched paths and produce the (normalized) symbol score S'_{sym}. For query formula Q and document formula D, the final symbol score factor we multiply is a rescaled version:

$$S_{sym}(Q, D \mid w^*) = \frac{1}{1 + (1 - S'_{sym})^2} \tag{3}$$

Second, we add penalties to long formulas in a document. Assume the original formula length is L_D, the penalty $P(D; \eta)$ is parameterized by $\eta \in [0, 1]$:

$$P(D; \eta) = 1 - \eta + \eta \cdot \frac{1}{\log(1 + L_D)} \tag{4}$$

Finally, the overall formula score for math formula similarity $S(Q, D)$ is given by

$$S(Q, D) = w^*(Q, D) \cdot S_{sym}(Q, D \mid w^*) \cdot P(D; \eta) \tag{5}$$

This scoring process is accelerated by the GBP-LEN dynamic pruning strategy [15].

To handle text keywords, we adopt the BM25+ scoring schema [49]. The overall score in Approach Zero is a weighted sum (using a *math path weight* [47] to weigh a math match over text-word match) of all partial scores obtained by BM25+ in normal text keywords and those by formula scoring in formula keywords.

3.2. ColBERT

The ColBERT model [4] is a dense retrieval model based on a BERT backbone. It is considered a bi-encoder model because it has two independent encoders, i.e., a query encoder and a document encoder. The interaction between them is deferred until the similarity scoring takes place, and this similarity is only dependent on their encoded embeddings and it is calculated using the MaxSim operation [4]. In addition, unlike other dense retrievers that use passage-level [CLS] embedding, the ColBERT model preserves all output embeddings associated with each token. However, to reduce space footprint and speed up indexing, ColBERT pools each BERT output into a smaller dimension embedding ($d = 128$ by default). Because each output embedding is pretrained for the MLM objective [35], they represent fine-grained contextualized semantics for individual tokens. This also makes ColBERT easier to visualize the contribution of similarity by tokens.

For a query in token sequence $q = q_0, q_1, ...q_l$ and a passage in token sequence $p = d_1, d_2, ...d_n$, the ColBERT model calculates either dot product or L2 distance for the token-level score $s(q_i, d_j)$ over the normalized output embeddings between the corresponding tokens $i \in [E(q)], j \in [E(p)]$ of the query and passage. The overall scoring of query q and passage p is conducted by the MaxSim operation which locates the maximum matched token d_j in the passage for each query token q_i, and then it sums over their token-level scores:

$$S(q, p) = \sum_{i \in [E(q)]} \max_{j \in [E(p)]} s(q_i, d_j). \tag{6}$$

During the training, a triple of query and a contrastive passage pair, i.e., (q, p^+, p^-) is fed to the model to optimize a pairwise cross-entropy loss. At encoding, the model always prepends an unused token [Q] or [D] to differentiate the encoding of a query or a passage. In practice, the authors also use *query augmentation* by rewriting the padding query tokens [PAD]s to [MASK] tokens before query encoding. This has been demonstrated to boost effectiveness and it gets rid of the need to mask query tokens in batch processing.

In order to perform end-to-end retrieval, the index of ColBERT needs to include all encoded passage tokens as well as document IDs and their lengths to locate the offset of passage tokens. And for efficient query processing, a two-stage retrieval is done: (1) An approximate nearest neighbors (ANN) search (e.g., using the product quantization method [50]) is first performed to filter a pool of top candidate tokens for each query token individually. (2) Then it locates unique documents associated with the top candidate tokens and loads their entire passage embeddings

into GPU for fast MaxSim operation. Notice that the above candidate selection stage comes at a cost, it has rendered the end-to-end retrieval an approximate version of what is originally defined in Eq. 6.

Our implementation of the ColBERT model and its backbone are based on the Hugging Face Transformer [51] package and we use the Faiss package [52] to do first-stage ANN with product quantization. We use dot product for token-level scoring in both reranking and end-to-end retrieval. To accommodate the memory limit in our experimental environment, we choose to split the index into multiple shards and load them sequentially into memory and GPU. We adjust the division of shards to make sure each shard will not generate candidate embeddings that exceed the capacity of memory and GPU.

4. Handling ARQMath Tasks

Except for minor corpus fix, format changes, and topic renewal, Task 1 and Task 2 remain mostly unchanged compared to ARQMath 2021. However, the ARQMath Lab 2022 adds another open-domain QA task (i.e., Task 3) where it re-uses the same topics from Task 1 but only one single answer should be returned for each topic (potentially through extraction or generation).

In the following, we will focus more on our newly introduced model ColBERT and our handling of Task 3. For the Approach Zero system, only changes to the previous-year system [47] are described in detail. Both the Approach Zero and the ColBERT model are described in the beginning of this section because they are applied uniformly to all the tasks. [4]

4.1. Approach Zero Changes

The version of Approach Zero we use in this ARQMath Lab is basically the same as what we have used in ARQMath 2021 [47]. We do notice a leap in effectiveness when we evaluate Approach Zero on previous-year tasks. However, except for a list of major engineering improvements below, the model remains unchanged.

- **Improved OPT:** This includes unwrapping parentheses if it is redundant and directly under a fraction operator. And we unify some malformed LaTeX tokens with their correct forms, such as `sin` and `\sin`.
- **Simplified OPT:** Merge sign node into symbol hash value, delete unnecessary ADD nodes, and fix inconsistent grammar reduction. (See Section 3.1)
- **Bug fix:** Fixed a bug that prevents many paths from being indexed.

In Section 5.2, we will further evaluate the impact of effectiveness as a result of these changes.

4.2. Training Data

For pretraining, we use a corpus made by ourselves. [5] It is made of 1.69M documents crawled from the MSE and the Art of Problem Solving community (AoPS) website. During training, we

[4] For reproducibility, our system pipelines and model checkpoints are made available: https://github.com/approach0/pya0/tree/arqmath3

[5] To download our raw corpus: https://vault.cs.uwaterloo.ca/s/G36Mjt55HWRSNRR

exclude MSE posts after 2018. Sentence pairs are generated and concatenated with a [SEP] token where sentences are extracted by the following strategy: We set an input threshold for *two* sentences to at most 1/4 of the maximum tokens of the Transformer 10% of the time, and the other 90% of the time we use the default maximum number of tokens in Transformer. Then the length threshold for the first sentence is randomly selected between 1 and the input threshold, the second sentence is selected by filling up the rest input space. Sentences are concatenated until it exceeds their length threshold for the first time, and are truncated if it exceeds the maximum number of tokens in the Transformer. Compared to our previous work [10], we have improved the sentence splitting algorithm to include fewer short and meaningless sentences.

For training ColBERT, we use the ARQMath corpus which contains questions, answers, the number of upvotes by users, and links to the accepted answer as well as duplicate questions. The training triples only sample question-answer pairs. In particular, we use the accepted answer, or the accepted answer in a duplicate question, or any answer posts receiving more than 7 upvotes for a question as positive answers; while we mine hard negatives from the corpus by sampling random answers related to the same tags of the question. Finally, we have extracted 607K triplets for ColBERT training.

4.3. Preprocessing for Math

In this ARQMath Lab, we follow a similar way to handle math tokens: First, we add LaTeX math tokens as additional vocabulary before further pretraining. This helps to reduce the number of tokens for the input of the Transformer, which further allows more information to be considered for assessing similarity. Second, the existing LaTeX lexer used in Approach Zero is utilized (by calling a Python binding PyA0 [53]) to determine the added vocabulary. This additionally reduces the newly added vocabulary size as Approach Zero focuses on semantically relevant tokens and ignores unimportant tokens such as color and spaces in LaTeX commands.

Different from our previous work [10], we have adjusted our lexer slightly and pretrained a new backbone: We used to create a new token for each number less than 2 digits and create an extra special token BIGNUM for any other number tokens. However, this prevents decimals to be meaningfully represented and it disables the Transformer to learn any differences in larger numbers. We modified the way to tokenize a number by using the CHARACTER scheme proposed by Nogueira et al. [54], this orthography is robust to handle various user-created content and it has shown to help certain simple arithmetic downstream tasks to accurately calculate numbers to the 5th digit.

4.4. Training

We further pretrain a bert-base model and then we fine-tune a ColBERT model based on it. In a recent work [10], we have demonstrated the benefits for downstream MIR tasks to create a further pretrained backbone. Therefore, we pretrain the bert-base checkpoints with math vocabulary using the MLM and NSP objectives [35]. Instead of treating math and text separately as seen in [45], we generate sentence pairs by splitting passages containing math tokens just like normal full text, this creates a more diverse mixture of text and math tokens, presumably covering more heterogeneous data distributions.

On top of our pretrained backbone, we train the ColBERT model directly using the triples described in Section 4.2. In addition, we mask out punctuation symbols in a sentence to make more space for meaningful tokens. We apply the training procedure described in Section 3.2 for the ColBERT model.

4.5. Task 1: Answer Retrieval

Task 1 is about retrieving answers relevant to a question in full text.

For the Approach Zero pass, similar to our system in ARQMath 2021 [47], we prepend each answer post with its original question text before indexing to improve recall. In addition, we switch from the Lancaster stemmer to mainly using the Porter stemmer as we observe minor effectiveness gain for the latter one in the up-to-date Approach Zero. Furthermore, we use manually extracted text and formula keywords [6] for Approach Zero in Task 1, this not only mimics the ad-hoc search queries but also avoids hurting the effectiveness of a system based on strict matches when a misleading keyword or formula in the original topic is used. However, our manual topic for Task 1 is available [7] for anyone who wants to compare to our results directly.

In contrast to the Approach Zero pass, we use the complete topic content as the input for the ColBERT model as it can automatically encode tokens for similarity and this aligns with the way it is trained.

Finally, we combine the two systems by linear interpolation using the best parameters based on our previous K-fold cross validation on ARQMath-2 [47]. We also use ColBERT to rerank a base run produced by Approach Zero with no stemmer because reranking has generated the highest precision in one of our previous evaluations [47].

4.6. Task 2: Formula Retrieval

Task 2 asks to retrieve formulas relevant to the specified topic formula in a corresponding Task 1 question, given the formula's context.

For formula retrieval, we directly query the specified formula without considering the context in the Approach Zero pass. Similar to our approach in ARQMath 2021 [47], we rewrite the topic manually [6] to ensure the LaTeX can be correctly parsed into a clear OPT. For example, we manually insert a comma in B.336 to separate two conditions in a set expression (the original expression uses a long space to separate them), and we replace the text mode "`m++`" to "`m+1`" so that our parser can handle the rare increment expression correctly. If not manually corrected, we believe these changes are going to be difficult for being automatically suggested so that every user-generated formula can be converted to construct a clear OPT. In total, we have manually refined 20 topics in Task 2 for ARQMath 2022, and our rewritten topics for Task 2 are also made available. [8]

We use two approaches in the ColBERT pass to handle Task 2. The first one does not consider the context of the query formula even though the ColBERT is trained from both math and

[6] Our only manual intervention occurs at keywords extraction from official topics, other phases are automatic.
[7] https://github.com/approach0/pya0/blob/arqmath3/topics-and-qrels/topics.arqmath-2022-task1-manual.txt
[8] https://github.com/approach0/pya0/blob/arqmath3/topics-and-qrels/topics.arqmath-2022-task2-refined.txt

surrounding text. Thus it only passes an isolated formula to both the query encoder and document encoder, considering only the visual appearance of formulas. On the other hand, the second approach (we name it colbert_ctx) adds the dependency to the query formula context in the following way: We use the query formula ID (i.e., `qid`) to identify the specified formula in the full-text question, and mask every other formula except the query formula by rewriting each one to the special `[MASK]` token. Then, we feed ColBERT the modified tokens of the entire topic question.

Similarly, we generate a few fusion runs to combine the two passes by linear interpolation using the best parameters which are the same as what we choose to use in Task 1. However, we do not generate reranked results in Task 2.

To make sure we do not return more than 5 visually distinct formulas for a topic, we simply index at most 5 document formulas. For the colbert_ctx method, we index the full-text embeddings of a document, each formula in the document will get indexed with its formula ID and the complete document, masking out all other formulas. To reduce the number of embeddings to be indexed, we only consider formulas with original string lengths greater than 2 or those in the visually distinct formula set of our Task 1 index. For other methods (except the colbert_ctx), we only index formulas.

4.7. Task 3: Open Domain Question Answering

4.7.1. Model Selection

There are many choices for handling the open-domain QA problem. Parametric generative models are trained with questions as input and it outputs question-answer pairs without access to external knowledge. Such models store the required knowledge in the model parameters. On the other hand, non-parametric models mostly adopt a *retrieve-and-read* framework: they first retrieve relevant documents from the corpus given a question, and then produce the final answer based on these documents [55]. We adopt non-parametric models, which require smaller model sizes and less training data.

In non-parametric models, there are generative or extractive readers. Generally speaking, generative readers either need much training data to transfer themselves to the target domain or at least need a few examples to familiarize themselves with the answer formatting. Unfortunately, we do not have a good amount of training data for MIR, furthermore, generative models tend to recall non-overlapping spans in the training data, which leads to non-logical answers that are not desired for math question answering. Therefore, we adopt extractive readers in this task.

4.7.2. Generating Candidate Answers

We use our Task 1 runs for producing a set of candidate answers. As a principle, we choose to base our answer on a single post because different answer posts together will introduce different notations and dialects which create difficulties to align them in the final answer.

We design three strategies for post candidates:

- **Original**: A straightforward way to narrow down answer posts is to take the top-1 result from Task 1, we will use this as our first strategy

- **Re-rank**: The next strategy is simply considering a larger set of top results – here we use the top-20 results from Task 1 – and select one of them using the methods in the next subsection (Section 4.7.3).
- **Re-map**: Another strategy to look at the problem is to trust the community: we first try to retrieve the most similar corpus question post given a topic question (through the methods we use in Task 1), then we pick the accepted answer post, if non-exists, the top voted answer post as our candidate answer post.

4.7.3. Snippet Selection

A candidate post might be either inappropriately long or exceeding the length limit imposed by the Task 3 requirements, therefore we add a snippet selection step based on the candidate answer post(s). We split the answer post(s) into sentences with the PyA0 sentence splitting utility which takes care of the punctuations and avoids cutting them in the middle of a LaTeX string. Then a window of varied sizes (from a minimum of 5 up to 10 sentences) will go through the beginning of a post to its end to select a combination of sentence spans to form candidate snippets.

It is worth noting the beginning of a post usually contains some conclusive answers, or it serves as a good start for smooth reasoning, therefore we also add a selection strategy that always starts the window from the beginning of candidate answer post(s), but removes the limit of the window size unless it hits the end of an answer post.

We will generate candidate snippets according to all the above strategies, at the meantime, we also filter out a snippet if it:

- is shorter than 20 tokens,
- is longer than 1200 characters,
- has an odd number of "$"s, meaning that some math delimiters are *very likely* not placed correctly (admittedly, this is only a heuristic rule).

Finally, we use the same ColBERT model to score the snippets and pick the top 1 snippet as the final answer. Lastly, we also double-check the produced answers manually, if there are cases the final selected snippet is still too short, or there is no candidate available, or the math delimiters are still incorrect but not detected by the odd-"$" checking, we will randomly copy an answer from a parallel run among our (manually selected) 5 best runs for submissions. As a result, we mark our Task 3 runs as manual runs.

5. Experiment

5.1. Setup

For Approach Zero, we uniformly apply the same configuration as we choose for the primary run in 2021 (See Table 1). Rather than exploring different configurations, we fix the Approach Zero pass for all three tasks in 2022.

We have further pretrained a bert-base checkpoint with 5.8 M sentence pairs extracted from the MSE and AoPS corpus. The pretraining takes 9 epochs with only ∼1100 added math

Table 1
Configuration for Approach Zero.

Math path weight	**Formula length penalty** η	**BM25+** (b, k_1)	**Symbol match points** (b_1, b_2)
2.5	0.3	0.75, 2.0	0.94, 0.9

Table 2
Effectiveness impact evaluated by the ARQMath-2 topics for major changes in Approach Zero since 2021. Changes are shown in chronological order. (see Section 4.1 for detailed descriptions)

Changes	**Checkpoint SHA1**	**NDCG'**	**MAP'**	**P'@10**
(Our 2021 run)	eae6690e	0.351	0.137	0.189
Improved OPT	14d311b2	0.365	0.174	0.216
Simplified OPT	77e3571b	0.372	0.176	0.227
Bug fix	35afeb50	0.374	0.178	0.231
Switch stemmer etc.	(up to date)	0.383	0.190	0.235

vocabulary on three A6000 GPUs using a batch size of 114. Based on our backbone, we train ColBERT for 7 epochs also using three A6000 GPUs but with a batch size of 48. Following Reusch et al. [45], we set the maximum number of tokens to 512 for both pretraining and finetuning. The training, inference, and scoring are all done in half-precision.

In all the experiments, we use the AdamW optimizer [56] with a weight decay of 0.01, and a fixed learning rate of 1×10^{-6}.

5.2. Approach Zero Improvements

We have investigated the impact of major improvements after our previously published system run, results are shown in Table 2. It shows the most impactful changes are representational – after two major OPT improvements, we boost the official MAP ranking metric and precision metric by at least 28% and 20% over the result we have reported in 2021 – this has demonstrated that the design of representation in our structure search method is crucial for effectiveness.

5.3. ARQMath-1 and 2 Results

Table 3 and 4 show the evaluation results on ARQMath-1 and 2 topics using our systems in 2022. We compare ours to the official runs of the most effective systems in ARQMath 2021. We exclude comparing to this-year systems on previous topics because other systems may train on previous labels. Some of these systems are briefly mentioned in Section 2, however, here we describe relevant runs shown in these tables.

Task 1 Runs: The TU_DBS_P primary run from the TU_DBS team uses an official ALBERT cross encoder which is further trained for 750k steps and fine-tuned for 125k steps on the ARQMath data pairs split by sentence [45]. The DPRL_RRF is a Reciprocal Rank Fusion (RRF) between a run relying on MSE upvote and the DPRL_QASim run produced by the QASim method [41]. The MathDowsers_P is the primary run generated from the Tangent-L system [24],

Table 3
Effectiveness evaluation for previous ARQMath Labs (**Task 1**). Top-5 most effective systems for the year 2021 are compared to ours. In addition to the official measurements, we have also reported the BPref metric as well as the average number of judged hits per topic.

Runs	ARQMath-1					ARQMath-2				
	NDCG'	MAP'	P'@10	BPref	Judged	NDCG'	MAP'	P'@10	BPref	Judged
Year 2021										
TU_DBS_P	0.380	0.198	0.316	0.208	49.6	0.377	0.158	**0.227**	**0.158**	82.6
DPRL_RRF	0.422	**0.247**	**0.386**	**0.257**	37.8	0.346	0.101	0.132	0.083	83.5
DPRL_QASim	0.417	0.234	0.369	0.242	37.8	0.388	0.146	0.193	0.135	83.4
MathDowsers_P †	**0.433**	0.191	0.249	0.178	72.0	**0.434**	**0.169**	0.211	0.145	105.7
A0-B60 †	0.359	0.170	0.255	0.174	43.9	0.351	0.137	0.189	0.118	81.3
Year 2022										
colbert ♭	0.375	0.177	0.266	0.179	45.5	0.349	0.139	0.224	0.144	67.6
a0none †♭	0.376	0.202	0.269	0.205	40.0	0.383	0.190	0.235	0.182	69.8
a0porter †	0.373	0.204	0.270	0.204	39.5	0.383	0.185	0.241	0.172	71.4
rerank_nostemmer	0.382	0.205	**0.322**	0.202	40.0	0.385	0.187	0.276	0.187	69.8
fusion_alpha02	0.455	0.243	0.309	0.233	51.2	0.443	0.217	0.266	0.197	87.9
fusion_alpha03	0.460	**0.246**	0.312	**0.236**	52.5	0.450	0.221	0.278	0.208	89.0
fusion_alpha05	**0.462**	0.244	0.321	0.235	53.9	**0.460**	**0.226**	**0.296**	**0.211**	91.0

"♭": unsubmitted runs, "†": using unsupervised method.

Table 4
Effectiveness evaluation for previous ARQMath Labs (**Task 2**). Top-5 most effective systems for the year 2021 are compared to ours. In addition to the official measurements, we have also reported the BPref metric as well as the average number of judged hits per topic.

Runs	ARQMath-1					ARQMath-2				
	NDCG'	MAP'	P'@10	BPref	Judged	NDCG'	MAP'	P'@10	BPref	Judged
Year 2021										
Tangent-S ‡	0.691	0.446	0.453	0.412	39.1	0.492	0.272	0.419	0.290	48.2
DPRL_CFTED	0.648	0.480	0.502	0.475	31.4	0.410	0.253	0.464	0.260	33.5
DPRL_ltrall	**0.738**	**0.525**	**0.542**	**0.495**	39.0	0.445	0.216	0.333	0.228	46.2
MathDowsers_P †	0.561	0.370	0.447	0.374	28.6	0.552	0.333	0.450	0.348	51.9
A0-P300 †	0.507	0.342	0.443	0.343	19.5	**0.556**	**0.361**	**0.488**	**0.361**	47.6
Year 2022										
colbert ♭	0.563	0.398	0.460	0.396	23.4	0.542	0.362	0.510	0.373	41.4
colbert_ctx ♭	0.256	0.168	0.252	0.188	7.4	0.228	0.120	0.308	0.135	11.4
approach0 †	0.582	0.446	0.477	0.455	22.9	0.573	0.420	0.588	0.417	37.1
fusion02_ctx	0.575	0.448	0.496	0.461	21.6	0.575	0.417	0.590	0.411	37.4
fusion_alpha02	0.633	0.502	0.513	0.502	26.4	0.646	0.469	0.597	0.457	48.6
fusion_alpha03	0.644	**0.513**	0.520	**0.510**	27.2	0.649	0.470	0.603	0.459	49.2
fusion_alpha05	**0.647**	0.507	**0.529**	0.501	27.1	**0.652**	**0.471**	**0.612**	**0.468**	50.1

"♭": unsubmitted runs, "†": using unsupervised method, and "‡": supervised only in reranking fusion.

it additionally captures repeated symbols and commutative operands based on manipulating the features extracted from the SLT representation, e.g., by enumerating the combination of all possible repeated symbol pairs and indexing all of them. The A0-B60 is our best previous-year submission for Task 1, produced by a combination of Approach Zero with Lucene/Anserini [47]. Our a0none and a0porter runs are Approach Zero runs using no stemmer and a Porter stemmer respectively. The rerank_nostemmer run is produced by reranking a0none using the ColBERT scorer. Finally, the fusion_alpha$*$ runs are a linear fusion interpolated by a convex combination where $\alpha = 0.2, 0.3$ and 0.5 respectively:

$$S_f = \alpha \cdot S_d + (1 - \alpha) \cdot S_a \tag{7}$$

where S_d, S_a are the scores produced from the dense retriever and Approach Zero respectively, and S_f is the fusion score.

Task 2 Runs: The Tangent-S [27] is a formula search engine using both SLT and OPT, it has been used as a baseline for formula search in ARQMath-1 and 2. Tangent-S makes no use of the question text in Task 2. The DPRL_ltrall reranks a list of 6 signals using SVM-rank [57], the training data includes all ARQMath-1 judgments (77 queries). The DPRL_CFTED reranks the results from Tangent-CFT using tree-edit distance, the latter learns fastText [29] n-gram embedding from SLT, SLT operators, and OPT structure features. The MathDowsers_P is generated by the same system from the MathDowsers team [23, 24] in Task 1, except it adds a strategy to rank a set of visually distinct formula candidates by utilizing their Task 1 result. Our approach0 run is the formula-only retrieval for Task 2 by the up-to-date Approach Zero system. The fusion02_ctx and fusion_alpha$*$ runs are the convex combination (See Equation 7) with the approach0 run by the colbert_ctx run ($\alpha = 0.2$) and the colbert run ($\alpha = 0.2, 0.3$ and 0.5) respectively.

5.4. ARQMath-3 Results

We present the results for ARQMath-3 in Table 5, 6, and 7. In this ARQMath Lab, a total of 9 teams have participated, and we achieve the best results in all three tasks. Here we briefly describe some of the top systems we are comparing in this paper. For a complete overview of other participant systems, please refer to Mansouri, Novotný, Agarwal, Oard, and Zanibbi [58].

Task 1 Baselines: According to Geletka et al. [59], the selected MSM run is produced from an ensemble model in which each method is mainly developed as part of an Information Retrieval course taught at the Faculty of Informatics, Masaryk University, Brno, Czech Republic. And the MIRMU run is produced by a dense retriever pipeline that uses a miniLM as a bi-encoder (first-stage) retriever and a RoBERTa model as a cross-encoder reranker.

Task 2 Baselines: The DPRL Tangent-CFTED uses tree-edit distance to rerank a set of candidates retrieved by formula FastText embeddings trained on structure features, see Section 2 for detail. And the latex_L8_a040 from MathDowsers is the default configuration for a newly rewritten and improved system on their previous Tangent-L system, with a relative weight of 0.40 on math tuples (over text terms).

Task 3 Baselines: In Task3, *text-davinci-002*, the most capable model of GPT-3 [60] is used as the baseline system. Another generative run *amps3_se1_hints* by the TU_DBS team uses

Table 5
Results for the ARQMath-3 (2022) main task. Our baseline and submission runs are compared to a selected set of the best results from other participating teams. We are ranked at the top in terms of all effectiveness metrics. In addition to the official measurements, we have also reported the BPref metric as well as the average number of judged hits per topic. Our baseline runs using structure search without learning on data can still be competitive among top systems.

Runs	ARQMath-3 Task 1				
	NDCG'	MAP'	P'@10	BPref	Judged
Others (team / run)					
MSM / Ensemble_RRF	0.504	0.157	0.241	0.138	154.9
MIRMU / MiniLM+RoBERTa	0.498	0.184	0.267	0.169	120.8
Ours					
colbert ♭	0.418	0.162	0.251	0.165	89.0
a0none †♭	0.397	0.154	0.262	0.160	77.8
a0porter †	0.397	0.159	0.271	0.164	76.9
rerank_nostemmer	0.418	0.172	0.309	0.189	77.8
fusion_alpha02	0.483	0.195	0.305	0.184	105.2
fusion_alpha03	0.495	0.203	0.317	0.192	107.6
fusion_alpha05	**0.508**	**0.216**	**0.345**	**0.207**	110.0

"♭": unsubmitted runs, "†": using unsupervised method.

the GPT-2 model [61] but is further fine-tuned on the AMPS dataset [31]. They additionally prepend the prompt word "HINT" to the beginning during decoding. On the other hand, the DPRL run, SBERT-SVMRank, uses an extractive approach based on SVM and Sentence-BERT models.

As required, we apply the same methods that generate our runs for Task 1 and Task 2 in all evaluations. For Task 3 in ARQMath-3, our methods are described in Section 4.7.2.

5.5. Effectiveness Discussion

Analysis: Our system has advanced substantially from 2021, this comes in with two folds, i.e., the Approach Zero improvements and the introduction of ColBERT: (1) The boost in Approach Zero scores is mainly attributed to our engineering improvements, this includes the enhancement of our OPT representation (see Section 3.1 and 4.1). These changes make our Approach Zero base runs alone very effective, especially in formula retrieval, outperforming other systems in almost every metric without using any training data. (2) Because this year we have introduced a strong dense retriever and it is demonstrated to be a complementary component to the structure search. Although combining a much more efficient DPR dense retriever [3] may still be very effective according to our previous study [10], we find the ColBERT model more effective and robust as it provides fine-grained contextualized embeddings.

Our implementation of the ColBERT is among the few bi-encoder dense retrievers being able to achieve effective results in the formula-centered retrieval task, i.e., Task 2, despite the fact that we are using the same ColBERT model from Task 1, trained on math QA posts with text around formulas. However, we notice the ranking on structure search does not provide such

Table 6
Results for the ARQMath-3 (2022) formula retrieval task. Our baseline and submission runs are compared to a selected set of the best results from other participating teams. We are ranked at the top in terms of all effectiveness metrics. In addition to the official measurements, we have also reported the BPref metric as well as the average number of judged hits per topic. Our baseline run using structure search without learning on data can also outperform other systems except for NDCG'.

Runs	ARQMath-3 Task 2				
	NDCG'	MAP'	P'@10	BPref	Judged
Others (team / run)					
DPRL / Tangent-CFTED	0.694	0.480	0.611	0.471	61.7
MathDowsers / latex_L8_a040 †	0.640	0.451	0.549	0.443	60.3
Ours					
colbert ♭	0.604	0.436	0.622	0.446	42.8
colbert_ctx ♭	0.152	0.080	0.218	0.093	6.4
approach0 †	0.639	0.501	0.615	0.505	45.9
fusion02_ctx	0.631	0.490	0.611	0.499	45.0
fusion_alpha02	0.715	0.558	0.659	0.553	55.3
fusion_alpha03	**0.720**	0.565	0.665	**0.562**	55.8
fusion_alpha05	**0.720**	**0.568**	**0.688**	0.560	56.2

"♭": unsubmitted runs, "†": using unsupervised method.

Table 7
Effectiveness evaluation for ARQMath-3 Labs (**Task 3**). The *Posts* column lists the candidate selection strategy described in Section 4.7.2. The *Start* column lists whether we select the answer sentence from the beginning of the answer posts or not, as described in Section 4.7.3. The *Type* column lists the type of the QA model: 'E' for extractive and 'G' for generative.

Runs	Base Run	Posts	Start	Type	AP'	P'@1
Others (team / run)						
GPT-3 (baseline)	-	-	-	G	**1.346**	**0.500**
DPRL / SBERT-SVMRank	-	-	-	E	0.462	0.154
TU_DBS / amps3_se1_hints	-	-	-	G	0.325	0.078
Ours						
run5	colbert	Re-map	middle	E	0.949	0.282
run4	rerank_nostemmer	Re-map	middle	E	1.115	0.321
run3	fusion_alpha05	Re-rank	middle	E	1.179	0.372
run2	fusion_alpha05	Original	beginning	E	1.231	0.397
run1	rerank_a0porter	Re-rank	middle	E	**1.282**	**0.436**

benefits as it is compared to fusion results with first-stage ColBERT search. This indicates that ColBERT provides fewer benefits in boosting the precision of a structure search system than it provides through adding recall to the base run.

Interestingly, the powerful GPT-3 model, i.e., text-davinci-002, can answer math questions very well, even better than our extractive approach based on a highly effective retriever. Although it is unclear to us whether this GPT-3 model simply recalls some holdout answers in the

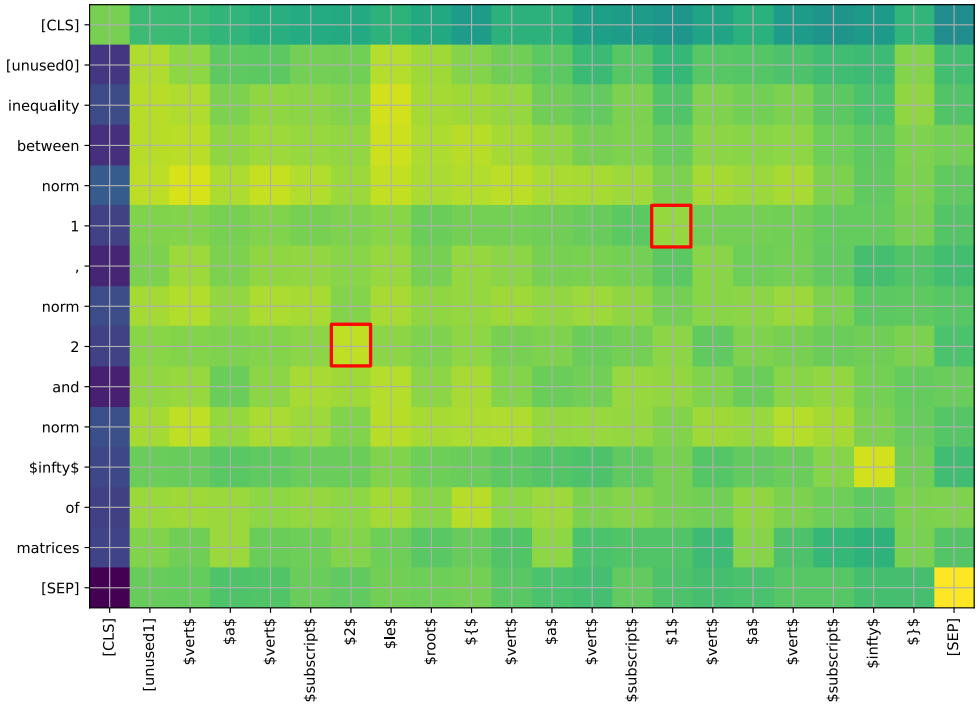

Figure 2: Visualization for the ColBERT token matching between two heterogeneous tokenized sentences extracted from an example topic (A.301). Tokens wrapped in dollar signs are those that originally appear inside math-mode LaTeX. The highlighted grids in red demonstrate the model is capable to link the same math entities from math mode and non-math mode.

ARQMath dataset from its training data, its performance is surprising to us.

Visualization: To illustrate the benefits that ColBERT can provide, we visualize its final scoring matrix during the MaxSim operation (See Figure 5.5). The heatmap in Figure 5.5 are ColBERT partial scores of each pair of tokens in two similar but heterogeneous sentences, i.e., "Inequality between norm 1, norm 2 and norm ∞ of matrices" and "$\|A\|_2 \leq \sqrt{\|A\|_1 \|A\|_\infty}$". The ColBERT model is able to (1) identify the same math entity even if they are from different modes, (2) handle formulas even if they are malformed and cannot be handled correctly by our existing parser, and (3) capture the similarity between relevant formulas even if they are structurally different. For example, it can associate text entity *inequality*, *norm*, and *matrices* to the correct "$\leq$", "$|$" math tokens, and the variable "a" (it actually associates to an uppercase "A" in the original topic, this is because the Huggingface tokenizer changes the input to lowercase by default. However, this is unintended behavior, we may need to fix it in the future). In addition, the [CLS] embedding may also capture high-level passage semantics. These strengths will offset some of the most important weaknesses in our structure search system.

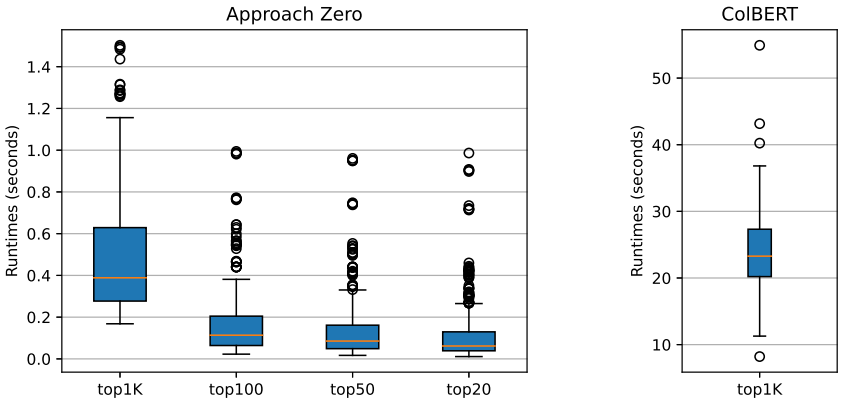

Figure 3: Query run times of Approach Zero and our implementation of ColBERT evaluated by the ARQMath-3 Task 1 topics. Approach Zero latencies are shown for different threshold values, and the distribution is averaged over 5 runs. In the ColBERT case, we need to load 14 shards into GPU memory subsequently for each query topic due to our GPU memory limit.

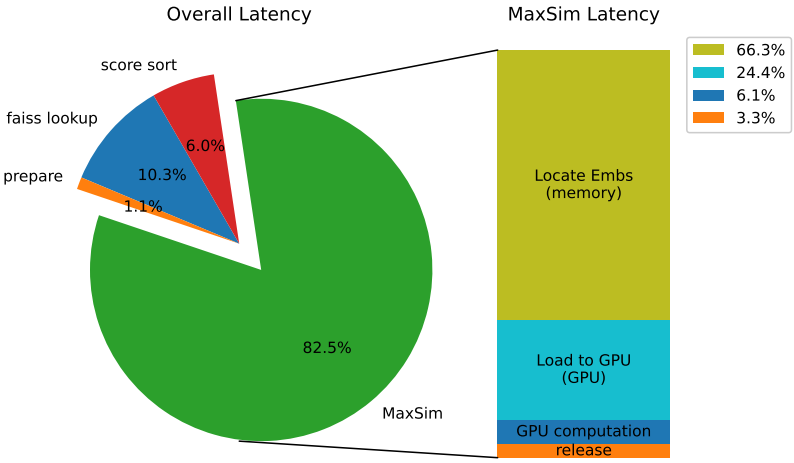

Figure 4: Latency decomposition for our ColBERT implementation. This graph only reflects single-shard latencies in the MaxSim operation. In practice, we need to load multiple shards into GPU when GPU memory is not sufficient to perform the matrix multiplication for all candidates.

5.6. Efficiency Discussion

We report our efficiency for both Approach Zero and our implementation of ColBERT in Figure 3. The Approach Zero run times are reported on a personal workstation with Intel Core i5-8600K CPU, 32 GiB memory, and Toshiba HDWD110 hard drive. The ColBERT is running on a server with Xeon(R) 4210 CPU, 370 GiB memory and A6000 GPUs. Both passes are running in a single-thread experimental environment.

As illustrated in Figure 3, the Approach Zero system is able to finish a query in a sub-second on average even if it is configured to return top-1000 results. Compared to a 3 times higher average run time we have reported in 2021 [47] where we run evaluation on a potentially heavily loaded shared server, we believe our evaluation this time on a personal workstation in a low workload environment reflects the efficiency more accurately. In addition, we find that the run times become much smaller after one "warm-up" run. To investigate this, we use the Linux **strace -cw** command to summarize the wall clock times for the query processing of the first 10 ARQMath-3 topics. We find that the first-time run spends 62.10% of the time on the *read* system calls which are invoked about 300K times in total and they mainly reflect the disk IO wait times. After the first run, only 16.46% of the time is spent on the read system calls. This indicates that the index is implicitly cached into memory (e.g., by the OS or filesystem) during the first run, and this first-time run could cause a high variance if not explicitly caching the index into memory. Because our run times are reported after running queries a few times, they should not be considered as entirely on-disk runs.

For the ColBERT model, because we use 512 maximum number of embeddings and the ARQMath topics contain a lot of math tokens and are generally long enough, it is suboptimal in efficiency. Furthermore, the way our ColBERT was initially implemented considers this large GPU memory consumption (quadratic in query length) and low memory resource constraints imposed on the cluster we were running. As a result, it has to load and spill multiple shards of the index in sequence *for each query* to cope with different resource limits, resulting in the notable inefficiency in our ColBERT pass. In our case, it takes more than 20 seconds to run a single topic on average, each loading 14 shards of the 312 million embeddings in total. In fact, the majority of the query processing in latencies is to locate candidates in the main memory and load them to GPU (See Figure 4). Obviously, the resource requirement and time cost for our current ColBERT model implementation are impractical. However, recent developments of ColBERT [7, 8] have shown the potential to lower the order of magnitude of the space footprint and query latencies. Additionally, other effective sparse retrieval systems based on learned dense representations [62, 63, 64] also show promising results. These could be the alternatives to be studied by us while keeping our system at the same effectiveness level.

6. Conclusion

It is shown that a structure-match search method, when combined with a dense retriever in an end-to-end pipeline, can be very effective in the math answer retrieval tasks. This is because the ColBERT model can discover related connections and overcome the issue of imposing too many lexical or structural constraints over search candidates. However, the need to keep multiple embedding vectors makes it expensive in resources. We believe an important direction to continuously advance MIR in the future is to efficiently and effectively capture semantic similarities lacking lexical agreements and, additionally, to capture math transformations beyond structure matching. Lastly, we are truly at the dawn of an exciting time where large language models like the GPT-3 and others [65, 66] keep surprising us with their capabilities. It remains to be seen whether a generative approach in the future can serve as a dominant role to directly and fully handle queries in the domain of math IR.

Acknowledgments

As ARQMath has come to an end in the CLEF Lab, we greatly appreciate task organizers and the National Science Foundation (NSF) for hosting and funding the CLEF ARQMath tasks for the past three years. Without these opportunities, we could not identify our weaknesses and advance our system continuously.

This research was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada. Computational resources were provided by Compute Ontario and Compute Canada.

References

[1] B. Mansouri, A. Agarwal, D. Oard, R. Zanibbi, Finding old answers to new math questions: The ARQMath Lab at CLEF 2020, in: J. M. Jose, E. Yilmaz, J. Magalhães, P. Castells, N. Ferro, M. J. Silva, F. Martins (Eds.), Advances in Information Retrieval, 2020. URL: https://www.cs.rit.edu/~rlaz/files/ARQMATH_Lab_overview_.pdf.

[2] B. Mansouri, R. Zanibbi, D. W. Oard, A. Agarwal, Overview of ARQMath-2 (2021): Second CLEF lab on answer retrieval for questions on math (working notes version), in: Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum, 2021. URL: http://ceur-ws.org/Vol-2936/paper-01.pdf.

[3] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, W. Yih, Dense passage retrieval for open-domain question answering, arXiv:2004.04906 (2020). arXiv:2004.04906.

[4] O. Khattab, M. Zaharia, ColBERT: Efficient and effective passage search via contextualized late interaction over BERT, in: SIGIR, 2020. URL: https://dl.acm.org/doi/abs/10.1145/3397271.3401075.

[5] L. Gao, J. Callan, Condenser: A pre-training architecture for dense retrieval, arXiv:2104.08253 (2021). arXiv:2104.08253.

[6] S. Hofstätter, S.-C. Lin, J.-H. Yang, J. Lin, A. Hanbury, Efficiently teaching an effective dense retriever with balanced topic aware sampling, in: SIGIR, 2021. URL: https://doi.org/10.1145/3404835.3462891.

[7] K. Santhanam, O. Khattab, J. Saad-Falcon, C. Potts, M. Zaharia, ColBERTv2: Effective and efficient retrieval via lightweight late interaction, arXiv:2112.01488 (2021). arXiv:2112.01488.

[8] K. Santhanam, O. Khattab, C. Potts, M. Zaharia, Plaid: An efficient engine for late interaction retrieval, arXiv:2205.09707 (2022). arXiv:2205.09707.

[9] J. Lin, A proposed conceptual framework for a representational approach to information retrieval, 2021. arXiv:2110.01529.

[10] W. Zhong, J.-H. Yang, J. Lin, Evaluating token-level and passage-level dense retrieval models for math information retrieval, 2022. arXiv:2203.11163.

[11] B. R. Miller, A. Youssef, Technical aspects of the digital library of mathematical functions, in: AMAI, 2003. URL: https://link.springer.com/article/10.1023/A:1022967814992.

[12] Y. Hijikata, H. Hashimoto, S. Nishida, Search mathematical formulas by mathematical

formulas, in: SHI (Symposium on Human Interface), 2009. URL: https://link.springer.com/content/pdf/10.1007/978-3-642-02556-3_46.pdf.

[13] K. Yokoi, A. Aizawa, An approach to similarity search for mathematical expressions using MathML, in: DML (Digital Mathematics Library), 2009. URL: https://dml.cz/handle/10338.dmlcz/702557.

[14] W. Zhong, R. Zanibbi, Structural similarity search for formulas using leaf-root paths in operator subtrees, in: ECIR, 2019. URL: https://par.nsf.gov/servlets/purl/10124342.

[15] W. Zhong, S. Rohatgi, J. Wu, L. Giles, R. Zanibbi, Accelerating substructure similarity search for formula retrieval, in: ECIR, 2020. URL: https://link.springer.com/chapter/10.1007/978-3-030-45439-5_47.

[16] R. Zanibbi, D. Blostein, Recognition and retrieval of mathematical expressions, in: IJDAR, 2012. URL: https://link.springer.com/article/10.1007/s10032-011-0174-4.

[17] T. Schellenberg, B. Yuan, R. Zanibbi, Layout-based substitution tree indexing and retrieval for mathematical expressions, in: DRR, 2012. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/8297/82970I/Layout-based-substitution-tree-indexing-and-retrieval-for-mathematical-expressions/10.1117/12.912502.short?SSO=1.

[18] N. Pattaniyil, R. Zanibbi, Combining tf-idf text retrieval with an inverted index over symbol pairs in math expressions: The tangent math search engine at ntcir 2014., in: NTCIR, 2014. URL: https://research.nii.ac.jp/ntcir/workshop/OnlineProceedings11/pdf/NTCIR/Math-2/08-NTCIR11-MATH-PattaniyilN.pdf.

[19] R. Zanibbi, K. Davila, A. Kane, F. Tompa, The Tangent search engine: Improved similarity metrics and scalability for math formula search, arXiv:1507.06235 (2015). arXiv:1507.06235.

[20] R. Zanibbi, K. Davila, A. Kane, F. Tompa, Multi-stage math formula search: Using appearance-based similarity metrics at scale, in: SIGIR, 2016. URL: https://dl.acm.org/doi/abs/10.1145/2911451.2911512.

[21] D. Fraser, A. Kane, F. Tompa, Choosing math features for BM25 ranking with Tangent-L, in: DocEng, 2018. URL: https://dl.acm.org/doi/abs/10.1145/3209280.3209527.

[22] F. Dallas, Math Information Retrieval using a Text Search Engine, Master's thesis, University of Waterloo, 2018. URL: https://uwspace.uwaterloo.ca/handle/10012/13329.

[23] Y. K. Ng, D. J. Fraser, B. Kassaie, G. Labahn, M. S. Marzouk, F. Tompa, K. Wang, Dowsing for math answers with Tangent-L, in: CLEF, 2020. URL: https://link.springer.com/chapter/10.1007/978-3-030-85251-1_16.

[24] Y. K. Ng, D. Fraser, B. Kassaie, F. Tompa, Dowsing for answers to math questions: Ongoing viability of traditional MathIR, in: CLEF, 2021. URL: http://ceur-ws.org/Vol-2936/paper-05.pdf.

[25] Y. K. Ng, Dowsing for Math Answers: Exploring MathCQA with a Math-aware Search Engine, Master's thesis, University of Waterloo, 2021. URL: https://uwspace.uwaterloo.ca/handle/10012/17696.

[26] G. Y. Kristianto, G. Topic, A. Aizawa, MCAT math retrieval system for NTCIR-12 MathIR task, in: NTCIR, 2016. URL: http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings12/pdf/ntcir/MathIR/04-NTCIR12-MathIR-KristiantoGY.pdf.

[27] K. Davila, R. Zanibbi, Layout and semantics: Combining representations for mathematical

formula search, in: SIGIR, 2017. URL: https://dl.acm.org/doi/abs/10.1145/3077136.3080748.
[28] B. Mansouri, D. W. Oard, R. Zanibbi, DPRL systems in the CLEF 2020 ARQMath lab, in: CLEF, 2020. URL: https://par.nsf.gov/servlets/purl/10198749.
[29] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, arXiv:1607.04606 (2017). arXiv:1607.04606.
[30] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, et al., Scaling language models: Methods, analysis & insights from training gopher, arXiv:2112.11446 (2021). arXiv:2112.11446.
[31] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, J. Steinhardt, Measuring mathematical problem solving with the math dataset, arXiv preprint arXiv:2103.03874 (2021).
[32] S. Peng, K. Yuan, L. Gao, Z. Tang, MathBERT: A pre-trained model for mathematical formula understanding, arXiv:2105.00377 (2021). arXiv:2105.00377.
[33] R. Zanibbi, A. Aizawa, M. Kohlhase, I. Ounis, G. Topic, K. Davila, NTCIR-12 MathIR task overview, in: NTCIR, 2016. URL: http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings12/pdf/ntcir/OVERVIEW/01-NTCIR12-OV-MathIR-ZanibbiR.pdf.
[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: NIPS, 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
[35] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, arXiv:1810.04805 (2019). arXiv:1810.04805.
[36] V. Novotnỳ, P. Sojka, M. Stefánik, D. Lupták, Three is better than one: Ensembling math information retrieval systems., in: CLEF, 2020. URL: http://ceur-ws.org/Vol-2696/paper_235.pdf.
[37] V. Novotnỳ, M. Štefánik, D. Lupták, M. Geletka, P. Zelina, P. Sojka, Ensembling ten math information retrieval systems, in: CLEF, 2021. URL: http://ceur-ws.org/Vol-2936/paper-06.pdf.
[38] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, arXiv:1908.10084 (2022). arXiv:1908.10084.
[39] S. Rohatgi, J. Wu, C. L. Giles, Psu at clef-2020 arqmath track: Unsupervised re-ranking using pretraining., in: CLEF, 2020. URL: http://ceur-ws.org/Vol-2696/paper_121.pdf.
[40] S. Rohatgi, J. Wu, C. L. Giles, Ranked list fusion and re-ranking with pre-trained transformers for arqmath lab (2021). URL: http://ceur-ws.org/Vol-2936/paper-08.pdf.
[41] B. Mansouri, D. W. Oard, R. Zanibbi, DPRL systems in the CLEF 2021 ARQMath lab: Sentence-BERT for answer retrieval, learning-to-rank for formula retrieval, in: CLEF, 2021. URL: http://ceur-ws.org/Vol-2936/paper-04.pdf.
[42] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, Q. Liu, TinyBERT: Distilling BERT for natural language understanding, 2019. arXiv:1909.10351.
[43] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, T. Wang, Ms marco: A human generated machine reading comprehension dataset, 2016. arXiv:1611.09268.
[44] A. Reusch, M. Thiele, W. Lehner, An ALBERT-based similarity measure for mathematical answer retrieval, in: SIGIR, 2021. URL: https://dl.acm.org/doi/abs/10.1145/3404835.3463023.
[45] A. Reusch, M. Thiele, W. Lehner, TU_DBS in the ARQMath lab 2021, CLEF, in: CLEF, 2021.

URL: http://ceur-ws.org/Vol-2936/paper-07.pdf.
[46] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, R. Soricut, Albert: A lite bert for self-supervised learning of language representations, 2019. arXiv:1909.11942.
[47] W. Zhong, X. Zhang, J. Xin, J. Lin, R. Zanibbi, Approach Zero and Anserini at the CLEF-2021 ARQMath track: Applying substructure search and BM25 on operator tree path tokens, in: CLEF, 2021. URL: http://ceur-ws.org/Vol-2936/paper-09.pdf.
[48] W. Zhong, A novel similarity-search method for mathematical content in LaTeX markup and its implementation, Ph.D. thesis, University of Delaware, 2015. URL: https://udspace.udel.edu/handle/19716/17656.
[49] Y. Lv, C. Zhai, Lower-bounding term frequency normalization, in: Proceedings of the 20th ACM international conference on Information and knowledge management, 2011, pp. 7–16.
[50] H. Jégou, R. Tavenard, M. Douze, L. Amsaleg, Searching in one billion vectors: Re-rank with source coding, in: IEEE ICASSP, 2011. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5946540.
[51] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, A. Rush, Transformers: State-of-the-art natural language processing, in: EMNLP, 2020. URL: https://aclanthology.org/2020.emnlp-demos.6.
[52] J. Johnson, M. Douze, H. Jégou, Billion-scale similarity search with GPUs, IEEE Transactions on Big Data 7 (2019) 535–547. URL: https://ieeexplore.ieee.org/document/8733051.
[53] W. Zhong, J. Lin, PyA0: A Python toolkit for accessible math-aware search, in: SIGIR, 2021. URL: https://dl.acm.org/doi/abs/10.1145/3404835.3462794.
[54] R. Nogueira, Z. Jiang, J. Lin, Investigating the limitations of transformers with simple arithmetic tasks, arXiv:2102.13019 (2021). arXiv:2102.13019.
[55] L. Liu, P. S. H. Lewis, S. Riedel, P. Stenetorp, Challenges in generalization in open domain question answering, CoRR abs/2109.01156 (2021). URL: https://arxiv.org/abs/2109.01156. arXiv:2109.01156.
[56] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv:1711.05101 (2017). arXiv:1711.05101.
[57] T. Joachims, Training linear svms in linear time, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006. URL: https://doi.org/10.1145/1150402.1150429.
[58] B. Mansouri, V. Novotný, A. Agarwal, D. W. Oard, R. Zanibbi, Overview of ARQMath-3 (2022): Third CLEF lab on Answer Retrieval for Questions on Math (Working Notes Version), in: Working Notes of CLEF 2022 - Conference and Labs of the Evaluation Forum, 2022.
[59] M. Geletka, V. Kalivoda, M. Štefánik, M. Toma, P. Sojka, Diverse semantics representation is king: Mirmu and msm at arqmath 2022, in: CLEF, 2022.
[60] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, NeurIPS (2020). URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.
[61] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models

are unsupervised multitask learners, OpenAI blog (2019). URL: https://openai.com/blog/better-language-models/.

[62] Y. Bai, X. Li, G. Wang, C. Zhang, L. Shang, J. Xu, Z. Wang, F. Wang, Q. Liu, Sparterm: Learning term-based sparse representation for fast text retrieval, 2020. arXiv:2010.00768.

[63] T. Formal, B. Piwowarski, S. Clinchant, Splade: Sparse lexical and expansion model for first stage ranking, arXiv:2107.05720 (2021). arXiv:2107.05720.

[64] T. Formal, C. Lassance, B. Piwowarski, S. Clinchant, SPLADE v2: Sparse lexical and expansion model for information retrieval, arXiv:2109.10086 (2021). arXiv:2109.10086.

[65] M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, et al., Show your work: Scratchpads for intermediate computation with language models, arXiv:2112.00114 (2021). arXiv:2112.00114.

[66] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, arXiv:2203.11171 (2022). arXiv:2203.11171.