Word2Course: Creating Interactive Courses from as Little as a Keyword

Sébastien Foucher[a], Damián Pascual[b], Oliver Richter[c] and Roger Wattenhofer
Distributed Computing Group, ETH Zurich, Gloriastrasse 35, Zurich, Switzerland

Keywords: Web Scraping, Natural Language Processing, Question Generation, Distractor Generation.

Abstract: In this work, we introduce a novel pipeline that enables the generation of multiple-choice questions and exercises from as little as a topic keyword. Hence, providing users the possibility to start with a study objective in mind and then automatically generate personalized learning material. The main contributions of this project are a scraper that can extract relevant information from websites, a novel distractor generation method that can make use of context and a technique to automatically combine text and questions into interactive exercises. Our novel distractor generation method was tested in a human survey which showed that the distractor generation quality is comparable to hand crafted distractors. The pipeline is built into a web application that lets users refine the results for each step, openly accessible at https://adaptive-teaching.com.

1 INTRODUCTION

With the rapid pace of discoveries and the large diversity of topics, it is not always possible to find up-to-date learning material that is adapted to students' existing knowledge and interest. Sometimes such material might not have even been created yet, as the subject is too recent or too specific. As a result, a large number of man-hours is put into organizing new information into courses. This high workload is a major bottleneck when it comes to personalized teaching or to the wide adoption of automated teaching assistants in the future of education.

In this work, we present a framework that tries to address this bottleneck by allowing a user to generate questions and exercises from as little as a topic description. For this purpose, we deploy several recently proposed models for natural language processing and develop an ensemble of methods to generate misleading answers for multiple-choice questions. We will refer to these misleading answer options as *distractors*. We streamlined the developed tool into an application that is easy to use and publicly available at https://adaptive-teaching.com.

This application provides a user with the possibility to start with a study goal in mind, and then automatically generate personalized learning material to achieve it. To ensure good quality, the user is free to refine the process at any stage of the pipeline. This includes control over the exact information sources used, the questions to ask, and finally the existence of hints in the form of multiple-choice answer options. The resulting questions are free to be exported as exams, flashcards, or as exercises that can be studied without prior knowledge about the topic. To summarize, our contributions include:

- A web scraper to search and extract relevant text.
- A model for context based distractor generation.
- A topic-agnostic, automatic course generation.

2 RELATED WORK

Automating various aspects of course material generation has been a long-standing interest of the research community, going from discovering student's interests and abilities (Guetl, 2008) to the dynamic assembling of courseware (Sathiyamurthy et al., 2012). Our work differs from many previous proposals in that we eliminate the necessity for an organized knowledge base. As a matter of fact, most previous approaches to adaptive course generation depend on structuring the various concepts and learning materials into a database and developing an intricate combination system that generates new courseware by reassembling these concepts, see e.g. (del Carmen Rodríguez-Hernández et al., 2020). One of the early works in this field (Vassileva, 1992) intro-

[a] https://orcid.org/0000-0002-6869-1798
[b] https://orcid.org/0000-0002-7018-5368
[c] https://orcid.org/0000-0001-7886-5176

Foucher, S., Pascual, D., Richter, O. and Wattenhofer, R.
Word2Course: Creating Interactive Courses from as Little as a Keyword.
DOI: 10.5220/0011064700003182
In *Proceedings of the 14th International Conference on Computer Supported Education (CSEDU 2022) - Volume 1*, pages 105-115
ISBN: 978-989-758-562-3; ISSN: 2184-5026

duced an architecture that allowed authors of content to store teaching materials in a database. That material can then be assembled dynamically to form courses based on students' learning history and a tutor's pedagogic oversight. While much progress has been made since its introduction (Karampiperis and Sampson, 2004; Dario et al., 2005), a weakness of these traditional approaches is the need for manually structuring the information for its use in the database. Consequently, this often results in limited diversity of available course types. More recent projects like MotorAI (del Carmen Rodríguez-Hernández et al., 2020) have addressed this concern by enabling the use of more readily available data sources like PDF documents and WikiData (Vrandečić and Krötzsch, 2014).

To further reduce the need for structure, we use web-scraping. This approach is similar to the one used by the "Flexible" plugin (Parahonco and Petic, 2021) that scrapes relevant websites to downloads HTML or PDF files. The work that, to our knowledge, is the most related to ours, is the smart course generator from VitalSource[1]. Similar to us, it relies on the content provided by the student to build an interactive learning environment consisting of a sequence of questions. However, VitalSource relies on fill-in-the-blanks questions, while here we go a step further and generate questions with and without multiple-choice options. Our proposed pipeline consists of a question and a distractor generator. The question generator is built based on previous work (Cheng et al., 2021), while the distractor generator is a contribution of this work as no existing solution worked in our setup. The task of distractor generation consists of providing misleading answer options to a multiple-choice question. Notable research in that field was conducted by Zhaopeng Qiu et al. (Qiu et al., 2020) who introduced the EDGE framework that is the current state-of-the-art on the RACE dataset (Lai et al., 2017) for reading comprehension. Moreover, Siyu Ren, et al. (Ren and Zhu, 2020) achieved very good results on human evaluation of cloze-style open-domain multiple-choice questions. A novel feature of our distractor generator with respect to previous is its ability to use the outputs of the different stages of our pipeline. As such, it can rely on much more context than distractors developed on datasets like RACE.

[1]https://get.vitalsource.com

3 COURSE GENERATION PIPELINE

Our proposed information processing pipeline is depicted in Figure 1. Our pipeline consists of 4 stages: First, an *input stage* converts supported source types into web pages. Next, the *scraping stage* processes the web pages to extract relevant text and tables. Third, the *question generation stage* uses the extracted texts and tables to generate multiple-choice questions. Finally, an optional *exercise generation stage* assembles the questions and sources into interactive exercises that can be directly started without prior knowledge of the study topic.

3.1 Input Stage

Our pipeline supports three different types of inputs. Namely *PDF Documents*, *Website URLs* and *Topic descriptions*. Topic descriptions consist of a topic title/keyword and an optional specification that can be helpful to further narrow the search.

Processing Documents and Websites. Both PDF Documents and Website URLs are transformed into a web page using a Firefox browser[2], before being handed to the next stage.

Processing Topic Descriptions. The topic title is used as a query and executed by a search engine to obtain Website URLs. For our purposes, we choose to use the Google search engine and limit the search results to the first page. We choose Google because of its popularity and ease of limiting the output languages to English. Our method, however, also works with any other search engine that provides a web page interface. The found website URLs are then processed in the same manner as described in the previous paragraph so that the next stage receives a list of loaded web pages.

3.2 Scraper Stage

The purpose of the scraper is to identify and extract the relevant text and tables from a website. It achieves this by first identifying the text elements in a page and then classifying them into relevant or irrelevant text passages based on their properties. Later pipeline stages rely heavily on the extracted text quality to produce questions and exercises. The text output from the scraper is therefore treated as a guess that can be

[2]We note that Firefox transcribes PDF documents to HTML pages by default.

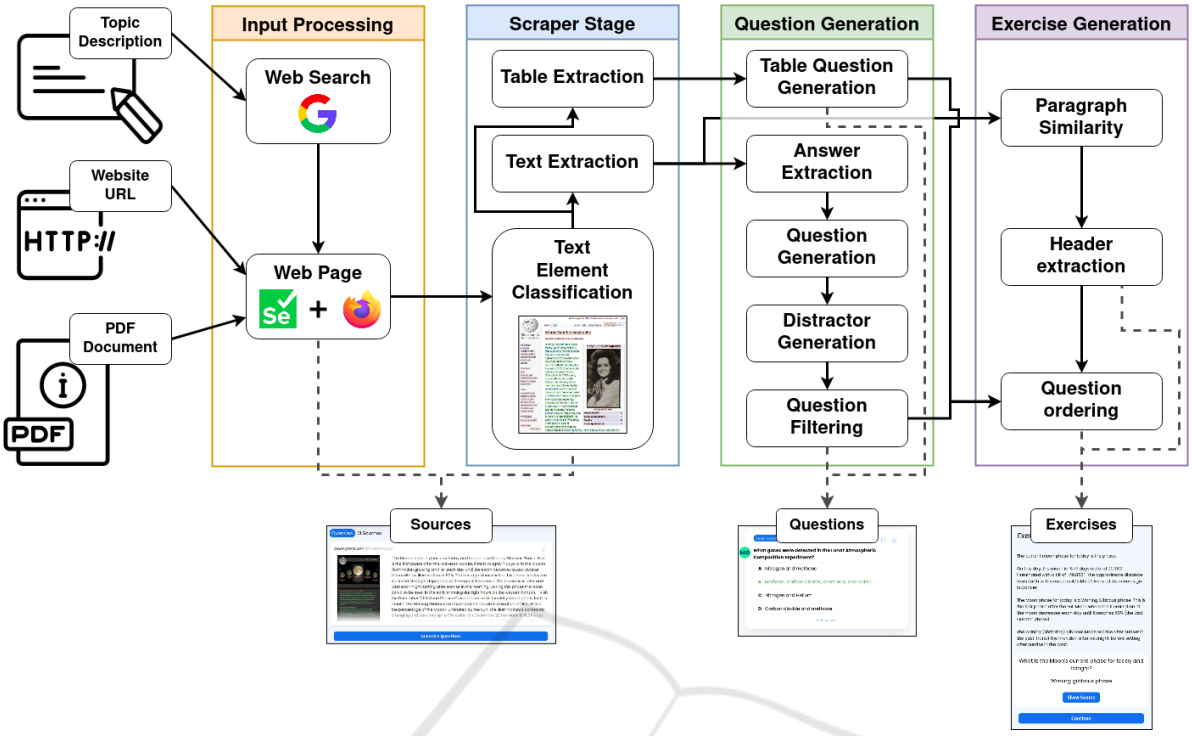

Figure 1: Diagram of the pipeline. The 4 stages are highlighted in orange, blue, green and purple respectively. Solid lined arrows denote a data flow inside the pipeline while dashed arrows denote exposure of the data in the application interface.

refined by the user through the application interface if necessary. We therefore built the scraper such that it visualizes the text that has been selected and gives the user the option to adjust the selection.

We select Selenium[3] as our scraping tool. It provides an automated browser environment in which it is possible to run dynamic content, interact with the loaded web pages as well as inject JavaScript code. A major factor behind this choice is the possibility to process PDF documents like websites without any overhead. While our scraper works for web pages of arbitrary finite length, the computational cost involved with text classification and later question generation from large texts made us constrain our page size to a maximum of 18000 pixels in length with a constant width of 720 pixels.

In order to communicate the text selection to the user, the scraper provides a screenshot of the web page over which the identified text boundary boxes are laid over. The text boundary boxes are obtained by accessing the lowest level rectangles in the Document Object Model (DOM) called `DOMRect`. Those rectangles only provide position and size information and are the result of processing the HTML code together with the CSS styling with respect to the window size. A difficulty we encountered was the presence of popup windows and backdrops that conceal the content of some pages, making it hard for the user to refine the text selection. The most reliable way we found to fix this issue required disabling the embedded JavaScript. The trade-off to this solution is that dynamic content will not be loaded, which can cause some websites to break.

[3]https://www.selenium.dev/

3.2.1 Text Extraction and Classification

Extraction starts by identifying the individual HTML elements that contain displayable text and returning them as a list. For each identified element we then determine properties like position, size, styling, and visibility by querying the browser. We then filter out elements that are set to be invisible and thus do not contain information a user would see.

A challenge we faced when building a classifier to label the relevant text elements was the lack of an existing dataset to train statistical models. We, therefore, handcrafted a small dataset and tested multiple models with different complexity.

Dataset. The dataset consists of 30 labeled web pages taken from 10 different domains (e.g., Wikipedia and the BBC News website are two different domains, while articles on these web sites are considered as web pages from these domains). Note

that web pages from the same domain tend to follow a similar Document Object Model (DOM) tree structure, therefore, the specification. Each page contains on average 510 labeled text elements, bringing the total number of labeled text samples to 15,291. For our evaluation, the 30 pages in the dataset were split into 20 training pages and 10 pages used for validation. The split was done such that the training set contained pages from 5 domains. Web pages from the same domains were also present in 5 of the pages in the validation set. The remaining 5 pages in the validation set were from new domains, which had a different overall page structure compared to the pages in the training set.

Models. We tested four different approaches for classification based on bounding box properties, DOM tree structure, and text cohesion. Bounding box properties are directly deduced from the `DOMRect` element and contain information about size and position. The (relative) location of each text element in the DOM Tree of the web page encodes further relevant information regarding the page layout surrounding an element. Finally, text cohesion can further help to classify individual text elements. We model text cohesion by iterating through all the text elements in the order of identification and filtering out the subset of elements that form cohesive sentences. Table 1 summarizes which properties are used by each of the models we tried. In particular, we compared the following approaches:

DOMRect Classifier. A gradient boosted decision tree provided by the CatBoost (Prokhorenkova et al., 2019) library. It uses the position and size properties of the text element bounding boxes.

DOM Tree Similarity (DTS). This approach keeps a database of labeled DOM subtrees from the training set. A new DOM tree is then labeled by finding the largest matching subtree in the database for each subtree of the DOM Tree.

Text Translator. Here we concatenate text elements together in the order they were identified. The resulting text is then processed by a T5 transformer which was trained to transcribe away all the irrelevant text.

Token Classifier (TC). Similar to the previous model, this model concatenates the list of text elements and feeds them to a T5 transformer. However, instead of transcribing the text, it classifies each individual token.

DTS + TC (Ensemble). As a final model, we took a soft voting ensemble consisting of the DOM Tree Similarity classifier and the Token Classifier.

Table 1: Properties used by the text extraction classifiers.

Model	Used properties		
	Bounds	**Tree**	**Text**
DOMRect Classifier	**Yes**	No	No
DOM Tree Similarity	No	**Yes**	No
Text Translator	No	No	**Yes**
Token Classifier	No	No	**Yes**
DTS + TC (Ensemble)	No	**Yes**	**Yes**

Table 2: Table showing the accuracy of the four models presented in Section 3.2.1. The first number gives the accuracy for same-domain web pages while the second number gives the accuracy for different-domain web pages.

Model	Accuracy
DOMRect Classifier	76% / 60%
DOM Tree Similarity (DTS)	**97% /65%**
Text Translator	57% / 57%
Token Classifier (TC)	54% / 55%
DTS + TC (Ensemble)	96% / 65%

Evaluation. The evaluation results shown in Table 2 are split according to the same-domains and different-domains parts of the validation set. As can be seen, the DOM Tree Similarity (DTS) classifier performed best in both same and cross-domain accuracy by a large margin. Approaches based purely on text coherence like the Token Classifier performed worst. In contrast to our expectations, the Token Classifier and DTS classifier did not complement each other very well and the DTS+TC ensemble did not improve over the baseline. We therefore went on with the DTS model.

Table Extraction. Apart from raw text we also extract tables from the processed web pages. To do so, we start by searching for all tables that contain at least one text element that was classified as relevant. We then extract the HTML table code and annotate rows and entries based on the relevance of the text entries they contain.

Similarity Computation. To facilitate refinement of the text element classification by the user, we compute a pairwise similarity measure between each pair of text elements. This measure is used to propagate a label adjustment by the user to similar elements, i.e., elements similar to the user-labeled one will change their label accordingly.

3.3 Question Generator

The generation stage takes the extracted text and tables from the scraping stage and produces questions.

Question generation from text was previously studied (Cheng et al., 2021). The question generation pipeline they introduced was shown to be capable of creating questions rivaling human ones in terms of correctness and comprehensiveness. To build on top of those findings, we reused the same answer extraction and question generation method but retrained the models on a larger dataset presented below. Our resulting pipeline supports three types of questions:

Open Response. Questions that allow respondents to answer in open text format so that they can answer based on their complete knowledge, feeling, and understanding.

Multiple Choice. Questions that have a list of answer options to choose from.

Table Filling. Questions that consist of a table with missing entries that need to be filled.

For Open Response and Multiple Choice questions we follow (Cheng et al., 2021) and first extract potential answers from the text and then generate questions conditioned on the extracted answers.

Answer Extraction. Answer extraction consists of finding excerpts in a text that could be a suitable answer to a question. This was achieved by fine-tuning a T5 (Raffel et al., 2020) model from Huggingface[4] on the dataset presented below. To achieve better control on the output quality, the input text was sequenced into sentences using the `PunktSentenceTokenizer` from the NLTK library. The model was then trained to take a sentence together with its context and to output the answer that was deduced from this sentence.

To account for the text diversity encountered when scraping arbitrary web pages we choose to use a combination of multiple datasets to train our model. We made use of the MRQA (Fisch et al., 2019) dataset which is already a combination of 18 existing Q&A datasets and we further included MOCHA (Chen et al., 2020), a dataset aimed at reading comprehension. The datasets were obtained from Huggingface[5] which already provides a train, validation, and test split. All our models that use this dataset followed the given split for their respective tasks.

Question Generation. Question generation takes an extracted answer and the corresponding context as input and tries to generate a fitting question. Similar to the answer extraction stage a pre-trained T5 model was used and fine-tuned on our merged dataset.

[4]https://huggingface.co/t5-base

[5]https://huggingface.co/datasets

3.3.1 Distractor Generation

Distractor generation is the stage responsible for providing misleading answer options for multiple-choice questions. We call those misleading answer options distractors. Three factors can be used to generate distractors. Namely: The generated question, the extracted answer, and finally the full text of the document used to generate the question. Note that the latter is not commonly accessible in distractor generation tasks based on datasets like RACE (Lai et al., 2017) and is a key factor that strengthens our environment. The distractor generation is achieved using an ensemble of different strategies. Table 3 lists all the strategies together with the properties they make use of.

When generating distractors each strategy returns a list of answer options. The answer options from all strategies are then aggregated, filtered, and sorted by quality using the distractor evaluator presented in Section 3.3.2. The top x results, where x is the number of desired distractors, are returned as final distractors.

T5 Closed-book. This strategy takes a question and tries to answer it without further information. The motivation behind this approach is that it could approximate the answers a respondent would give without reading the source text. The strategy is implemented by a T5 model that was fine-tuned on our merged dataset.

In early experiments we noted that this model tends to be biased towards certain answer formulations like starting answers with “just”, “nearly” or “about”. This bias is harmless when considering the answer separately, but when combined with the extracted answer and the remaining answer options the result tends to stand out. This makes it easy for respondents to spot the distractors generated by this model. We, therefore, added an additional filter that removes certain prefixes and answer formulations.

Entity Similarity. In this strategy, we made use of Named Entity Recognition (NER) to detect entities in the answer and replace them with similar entities from the source document. The spaCy[6] library is used to detect the entities in the answer and source document. The library also classifies the entities in types like Person, Organization, Date, Cardinal, etc.[7] The entities detected in the answer are then replaced uniformly at random with an entity from the source document that has the same class.

[6]https://spacy.io/

[7]You can find a complete list here: https://spacy.io/api/data-formats#named-entities

Table 3: Table showing what information is used by each of the distractor generation models.

Model	Used information		
	Question	Answer	Context
T5 closed-book	Yes	No	No
Entity similarity	No	Yes	Yes
Sense2vec	No	Yes	No
Sense2vec sub.	No	Yes	No
Semantic Net.	No	Yes	No

Sense2vec. This strategy makes use of Sense2vec (Trask et al., 2015) word vectors to suggest similar answer options. Sense2vec is an extension of the famous word2vec (Mikolov et al., 2013) algorithm that adds context sensitivity to the word embeddings. Finding a similar answer option consists of a x nearest neighbor search in the embedding space, where x is the number of desired distractors.

In contrast to our expectation, this strategy rarely creates synonyms and the ones that are created are removed at the filtering stage presented in Section 3.3.2. A weakness of this strategy is that generation is not possible for answers that have no direct word embedding.[8] To alleviate this issue we also introduced the strategy below.

Sense2vec Substitution. This strategy makes use of Named Entity Recognition (NER) to detect entities in the answer and then uses Sense2vec (Trask et al., 2015) to replace them with similar concepts. This makes it possible to apply the previous strategy to arbitrary answers. We want to emphasize, that this is not as general as the Sense2vec strategy, as some answers are not entities and will thus not be covered by this strategy. In practice, the extracted answer length strongly affects the quality of the entity recognition. As a result, this strategy struggles with answers consisting of two words or less. It can, therefore, be seen as a complement to the previous strategy.

Semantic Network. This strategy relies on extracting entities from the answer and using a semantic network to substitute them with related ones. For our pipeline, we used WikiData (Vrandečić and Krötzsch, 2014) as the semantic network. To find a similar entity, we first query the properties of the extracted entity and then initiate a second query to find different entities that share some of those properties.

[8]In practice, about 45% of the extracted answers can not be processed by this method.

3.3.2 Distractor Evaluation

The distractor evaluation stage aims to select the best distractors from the list of generated answer options obtained at the previous stage. It enables the selection of distractors based on the strengths of each distractor generation strategy. The stage has three steps.

The first step consists of filtering out all redundant answer options. An answer option is considered redundant if deleting all but alphanumeric characters results in the same string.

The second step evaluates how well each answer option fits the generated question. The evaluation is done using a binary classifier that was trained to distinguish correct answers from distractors based solely on the question and the answer option itself. The training was done on the SQuAD (Rajpurkar et al., 2016) dataset together with answer options generated using the previous stage. The model selected for classification is a case sensitive DistilBert (Sanh et al., 2020) model with pre-trained weights from Huggingface[9]. We note that for our pipeline we repeated the training step three times. Each time we used the distractors obtained after the updated evaluation stage.

Once the classifier provides a score for each answer option, we use a threshold to filter out unlikely candidates. The score ranges from 0 to 1 with 1 describing the better fit. The threshold was set to 0.5% and was determined using a visual inspection of 100 questions but was not tuned as no dataset about distractor quality was available at the time.

The last step consists of computing the similarity between each answer choice and removing distractors that are too similar, such that a minimum dissimilarity threshold is met. An answer choice can be either a distractor or the extracted answer. This step is achieved by first embedding each answer choice into a 768-dimensional vector and then computing the pairwise cosine similarities. To compute the embedding we use a pre-trained sentence BERT transformer (Reimers and Gurevych, 2019). The task of removing distractors that are too close is then equivalent to finding a vertex cover , where vertices represent answer choices that are connected by edges if a similarity threshold is exceeded. In practice, we set the maximum allowed cosine similarity to be 0.85.

After making sure that all thresholds are met, the remaining distractors are ordered according to their question matching score and only the top x distractors are returned, where x is the number of desired distractors.

In practice, we observed that some questions have less than x distractors which is due to the thresholds

[9]https://huggingface.co/distilbert-base-cased

we set and the tendency of some generators, like the T5 closed-book generator, to lack diversity in their output.

3.3.3 Table Question Generation

Table question generation consists of using the tables given by the scraper and generating questions about them. A great challenge we faced was the diverse usage of tables in web pages, making it often difficult to correctly interpret them. To solve this issue we decided to transform the task of question generation from tables to table filling. The benefit of this approach is that the interpretation of the table itself is deferred to the respondent.

3.3.4 Question Filtering

After generating multiple diverse questions, a question filtering stage is responsible for removing questions that are too similar to each other. It consists of first filtering out questions that use the same formulation or distractors and then making sure that the remaining questions meet a minimum dissimilarity threshold. Similar to the last step of the distractor evaluation, presented in Section 3.3.2, the question formulations are first embedded using pre-trained sentence BERT transformer (Reimers and Gurevych, 2019). Then the cosine similarity is computed and questions are removed using the vertex cover approach. Ties are resolved by keeping the question with the longest extracted answer.

3.4 Exercise Generation

The exercise generator aims at enabling a study environment that teaches the content from the sources and questions previously obtained from the question generation stage. The study environment consists of a step-wise introduction of paragraphs together with an occasional question to practice the newly acquired knowledge. The key factor needed to enable this step-wise study is the generation of a pairing between the context read by the user and the shown question. In this section, we are going to introduce a method to combine the different sources and to determine a suitable question based on similarities between source paragraphs.

3.4.1 Source Coupling

The idea behind the source coupling is to transform the source texts into a sequence of paragraphs while keeping track of potential information redundancies that can be used to select questions. The information redundancies will be encoded in the form of a paragraph similarity. Furthermore, to enable maneuvering to specific subjects, we also provide titles for each subtopic.

Instead of working on the raw extracted source text like the question generator, we instead use the extracted paragraphs that are provided by the scraping stage presented in Section 3.2. Those paragraphs can be seen as information containers. To determine if two of those containers share information, we compute a pairwise similarity measure between each pair of paragraphs.

The pairwise similarity is computed by first computing an embedding for each paragraph and then calculating the cosine similarity. To compute the embedding of a paragraph we start with extracting the sentences using the `PunktSentenceTokenizer` from the NLTK library. Each sentence is then embedded into a 768-dimensional vector by using a pre-trained sentence BERT transformer (Reimers and Gurevych, 2019), the same transformer we used to compute the answer similarities before. The paragraph embedding is then obtained by adding the vectors together and normalizing them to unit length.

To facilitate the browsing of topics in the list of paragraphs, we group consecutive paragraphs together and assign them a title. Titles are found by loading the HTML code where a paragraph was extracted from and retrieving the list of headings above that paragraph. Paragraphs that share the same heading are then grouped together.

When concatenating the source texts we keep the order of paragraphs inside the sources but we order the sources by specificity so as to start with the source that gives the broadest overview over a topic. The specificity of a source is determined by the similarity it shares with other sources. An assumption that is made is that less specific information is more likely to also occur in different source texts. Therefore, we decided to measure the specificity of a source by averaging the pairwise similarities of all paragraphs inside the source with paragraphs from all other sources. Note that this can be done efficiently using our paragraph similarity introduced above. Once the order is determined we finally concatenate the paragraphs from all the sources together. We thereby split the paragraphs into two groups: One that is shown to the user, and one that contains paragraphs with a high similarity to the ones that are shown. The second group will not directly be shown to the user, but is still used for exercise generation as discussed next.

3.4.2 Question Selection

An exercise is built such that after a few paragraphs have been presented to the user, a question is asked. The questions asked are thereby selected from the ones previously generated.[10] Before the selection process can begin, each paragraph needs to reference the questions that were generated from it. We then determine two groups of candidate questions. Group A consists of all the questions referenced by seen paragraphs and Group B consists of questions referenced by paragraphs that share strong similarity with at least one of the seen paragraphs. In practice, strong similarity is defined as a cosine measure larger than 0.5. Selecting a question then consists of randomly choosing one group. If it is group A we preferably choose a question from a paragraph seen longer than 3 minutes ago. For group B on the other hand, we choose a question from a paragraph that shares a similarity with the last paragraph the user read. In practice, we set the probability to choose group B over group A to 75%. In this setup the user gets iteratively asked about the information that he/she just read. While in some cases questions from group B cannot be answered with the information presented to the user so far, we see this as a feature rather than a bug as it encourages the user to actively think about the topic.

4 USER STUDY

Apart from the quantitative evaluations presented so far, we also investigate the quality of our pipeline presented in Section 3.3. As the answer extraction and question generation stage have previously been studied (Cheng et al., 2021), our evaluation is going to focus on the distractors.

This evaluation is conducted as survey where participants were shown the source text used for generation, the multiple-choice questions, and the sentence where the answer was extracted from. The exact layout of the survey can be found at https://adaptive-teaching.com/evaluation.

The survey was conducted with 40 participants using Amazon's Mechanical Turk[11]. Each participant had to evaluate 30 questions. The 30 questions were split into 20 questions generated by our pipeline, and 10 questions taken from the ARC dataset (Clark et al., 2018) which contains grade-school level, multiple-choice science questions. The choice of the dataset was motivated by the need to have questions resembling the ones generated by our pipeline while also providing multiple answer options and a source text. While the original ARC dataset does not provide the latter, there exists a version augmented with context sentences[12]. To reduce volatility, each question was evaluated separately by 5 participants, bringing the total amount of evaluated questions to 240 with 160 generated ones as opposed to 80 hand-crafted questions from the ARC dataset. The choice for this uneven split is motivated by our goal to also evaluate our 5 different distractor generation strategies presented in Section 3.3.1.

Table 4: Evaluation questions to be answered for each multiple-choice question. The scale is obtained by linearly mapping the answer options to an interval between 1 and 5 or 1 and 3 respectively. We recall that an answer option refers to either a distractor or the extracted answer.

Question	Answers
How would you rate the quality of the question?	Very Bad, Bad, Acceptable, Good, Excellent
How would you rate the quality of each individual answer option? *(Question is answered separately for each answer option)*	Very Bad, Bad, Acceptable, Good, Excellent
Can the question be answered using the highlighted text? *(Refers to the sentence the answer was extracted from)*	Yes, Yes partially, No
Is the highlighted answer option correct? *(Refers to the extracted answer)*	Yes, Yes partially, No

Before evaluating each question the participant was asked to read the text excerpt that was used to generate the question. The evaluation itself consisted of the four questions listed in Table 4. To answer the last two questions, we highlighted the sentence that the answer was extracted from.

4.1 Question Similarity

To ensure that our distractor evaluation is not confounded by discrepancies between the generated and hand-crafted questions themselves, we evaluated question quality, question answerability, and answer

[10]The application allows the user to also review generated questions before creating the exercise.

[11]https://www.mturk.com/

[12]https://www.tensorflow.org/datasets/catalog/ai2_arc_with_ir

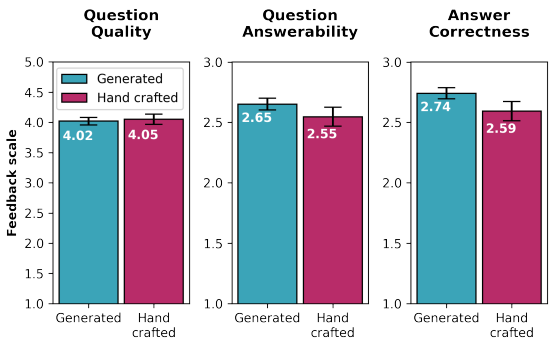

Figure 2: Comparison of generated (turquoise) and hand-crafted (cerise) questions in terms of question quality, question answerability and answer correctness. All metrics were evaluated by the subjects of the study. Feedback scores were obtained by mapping the user evaluation options to a linear scale and averaging them. The error bars show the 99% confidence interval of a t-distribution.

Figure 3: Evaluated quality comparison between generated (turquoise) and hand-crafted (cerise) distractors. The plot on the left shows the fraction of distractors assigned to each evaluation option. The plot on the right shows the average feedback score obtained by linearly mapping the user evaluation to a scale from 1 (Very Bad) to 5 (Excellent). The error bars on the right show the 99% confidence interval of a t-distribution.

correctness. A comparison of those three averaged metrics can be seen in Figure 2. While no significant difference in question quality was observed between the two sources, the generated questions had an edge over the hand-crafted ones both in terms of question answerability and answer correctness. We explain this difference by the fact that the source text for the ARC dataset was obtained by searching for an excerpt that answers the question instead of generating the question based on the source text. However, we do not believe that this difference is significant enough to have a major impact on the upcoming results.

4.2 Distractor Quality

The distractor quality was evaluated on a scale from 1 to 5 where 1 represents the worse end of the spectrum. As can be seen in Figure 3 the overall quality of the hand-crafted distractors was superior by a margin of 0.26 on the feedback scale. While the fraction of distractors with excellent and acceptable quality is very similar between the two sources, our generation process tends to produce fewer good distractors in favor of very bad ones. We explain this behavior by reminding that only 1 of the 5 distractor generators presented in Section 3.3.1 make use of the question context. Therefore, it is to be expected that some distractors make little sense in the true context of the question. Still, the result suggests that the overall quality of the generated distractors is surprisingly close to the quality of hand-crafted ones.

Next, we juxtapose the quality evaluation of distractors and answers. Figure 4 shows the qualities of generated distractors compared to extracted answers and Figure 5 does the same for the hand-crafted answer options. As can be seen, the quality gap between generated distractors and extracted answers is a significant 0.74 on the feedback scale. From the distribution, it is visible that 72.44% of the extracted answers are rated good or better, while only 48.01% of the generated distractors get the same rating. A less significant discrepancy can be seen on the human-crafted answer options. There the difference on the feedback scale is only 0.43.

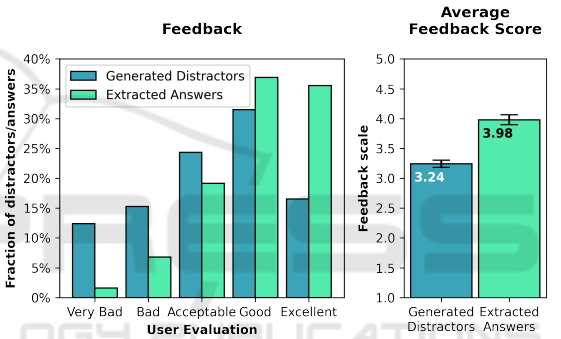

Figure 4: Evaluated quality comparison between generated distractors (turquoise) and extracted answers (green). The plot on the left shows the fraction of distractors/answers assigned to each evaluation option. The plot on the right shows the average feedback score obtained by linearly mapping the user evaluation to a scale from 1 (Very Bad) to 5 (Excellent). The error bars on the right show the 99% confidence interval of a t-distribution.

4.2.1 Distractor Quality across Generators

Next we compare the distractor quality across the 5 generators presented in Section 3.3.1. As can be seen in Figure 6, all generators performed very similarly on average. However, when comparing the feedback

Figure 5: Evaluated quality comparison between handcrafted distractors (cerise) and answers (melon). The plot on the left shows the fraction of distractors/answers assigned to each evaluation option. The plot on the right shows the average feedback score obtained by linearly mapping the user evaluation to a scale from 1 (Very Bad) to 5 (Excellent). The error bars on the right show the 99% confidence interval of a t-distribution.

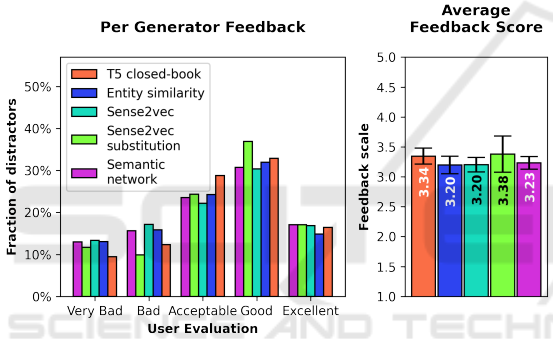

Figure 6: Evaluated quality comparison between distractor generators. The plot on the left shows the fraction of distractors assigned to each evaluation option. The plot on the right shows the average feedback score obtained by linearly mapping the user evaluation to a scale from 1 (Very Bad) to 5 (Excellent). The error bars on the right show the 99% confidence interval of a t-distribution.

distributions, there are some interesting differences. Notably, Sense2vec and Entity similarity performed very similarly across the spectrum even though they use very different replacement strategies. In contrast, Sense2vec substitution has the most distinct distribution and even produced 54.06% of distractors with a good or better rating. This is on par with the handcrafted distractors where 53.69% have such a rating. However, the Sense2vec substitution model was only responsible for 3.9% of the total amount of distractors. This is caused by the limitation of only handling answers where named entity recognition is possible (See Section 3.3.1 for more details).

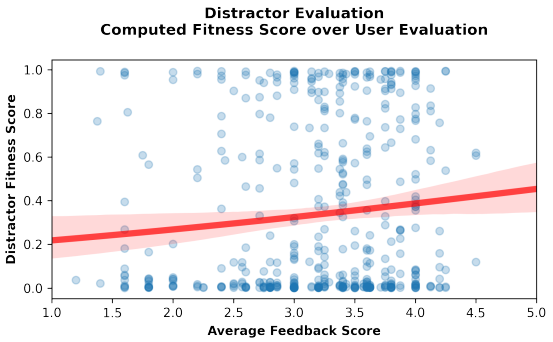

Figure 7: Comparison of the computed distractor fitness with the user evaluation. The distractor fitness is computed using the question matching model presented Section 3.3.2. Each blue marker represents a distractor. The average feedback score is computed by taking the mean result over all users who evaluated that distractor. The red line shows a logistic regression curve with a 95% confidence interval.

4.2.2 Question Matching Score

Finally, we also evaluate if the question matching score presented in Section 3.3.2 is well correlated with the perceived question quality. Unfortunately, as seen in Figure 7, that correlation is very weak. This is partially due to the fact that the model tends to be very partial with 68% of the samples scoring either below 10% or above 90%. These results can be explained in that the question matching model can rely on other queues than the actual information content to distinguish between distractors and answers. One of those possible queues could be the distractor formulations that are different from those of the correct answers. In particular, the lack of diversity in the distractor generator output can make it possible for the question matching model to remember how a distractor is typically formulated. We note that during development, a lack of diversity was noted in both the T5 closed-book and Semantic network generators.

5 CONCLUSION

We presented an end to end system to go from query keywords to full, interactive courses on a topic. Our evaluation shows that our pipeline is capable of producing distractors with a quality reasonably close to human-crafted ones, with no significant difference in the quality of our 5 distractor generators. However, there is room for improvement on the ability of the distractors to fool the user. One way to do so is to use more context in the distractor generators.

We deem our system as a valuable step towards a fully operational tool to automatically generate and

structure learning materials. Given the significant savings that such systems can represent in terms of time and effort from education professionals, we are convinced that they will become the angular stone of future education tools and we thus encourage further work on this topic.

REFERENCES

Chen, A., Stanovsky, G., Singh, S., and Gardner, M. (2020). Mocha: A dataset for training and evaluating generative reading comprehension metrics. In *EMNLP*.

Cheng, Y., Ding, Y., Foucher, S., Pascual, D., Richter, O., Volk, M., and Wattenhofer, R. (2021). Wikiflash: Generating flashcards from wikipedia articles. In Mantoro, T., Lee, M., Ayu, M. A., Wong, K. W., and Hidayanto, A. N., editors, *Neural Information Processing*, pages 138–149, Cham. Springer International Publishing.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Dario, N., Duque, N., Jimenez, C., ALBERTO, J., and LUNA, G. (2005). Ia planning for automatic generation of customized virtual courses.

del Carmen Rodríguez-Hernández, M., de la Vega Rodrigálvarez-Chamarro, M., Vea-Murguia Merck, J. I., Ballano, A. E., Lafuente, M. A., and del Hoyo-Alonso, R. (2020). Motoria: Automatic e-learning course generation system. In *Proceedings of the 2020 6th International Conference on Computer and Technology Applications*, ICCTA '20, page 92–96, New York, NY, USA. Association for Computing Machinery.

Fisch, A., Talmor, A., Jia, R., Seo, M., Choi, E., and Chen, D. (2019). MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *Proceedings of 2nd Machine Reading for Reading Comprehension (MRQA) Workshop at EMNLP*.

Guetl, C. (2008). Moving towards a fully automatic knowledge assessment tool. *International Journal of Emerging Technologies in Learning*, 3.

Karampiperis, P. and Sampson, D. (2004). Adaptive instructional planning using ontologies. pages 0–.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. (2017). Race: Large-scale reading comprehension dataset from examinations.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Parahonco, A. and Petic, M. (2021). Generation and use of educational content within adaptive learning. In *Workshop on Intelligent Information Systems*, volume 1, page 156.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2019). Catboost: unbiased boosting with categorical features.

Qiu, Z., Wu, X., and Fan, W. (2020). Automatic distractor generation for multiple choice questions in standard tests.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text. *arXiv e-prints*, page arXiv:1606.05250.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ren, S. and Zhu, K. Q. (2020). Knowledge-driven distractor generation for cloze-style multiple choice questions.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Sathiyamurthy, K., Geetha, T. V., and Senthilvelan, M. (2012). An approach towards dynamic assembling of learning objects. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, page 1193–1198, New York, NY, USA. Association for Computing Machinery.

Trask, A., Michalak, P., and Liu, J. (2015). sense2vec - a fast and accurate method for word sense disambiguation in neural word embeddings.

Vassileva, J. (1992). Dynamic courseware generation. 5:87–102.

Vrandečić, D. and Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.