AI-based Question Answering Assistance for Analyzing Natural-language Requirements

Saad Ezzini*, Sallam Abualhaija*, Chetan Arora‡§, Mehrdad Sabetzadeh†
*SnT Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
‡Deakin University, Geelong, Australia
§Monash University, Victoria, Australia
†School of Electrical Engineering and Computer Science, University of Ottawa, Canada
Email: {saad.ezzini, sallam.abualhaija}@uni.lu, chetan.arora@monash.edu, m.sabetzadeh@uottawa.ca

***Abstract*—By virtue of being prevalently written in natural language (NL), requirements are prone to various defects, e.g., inconsistency and incompleteness. As such, requirements are frequently subject to quality assurance processes. These processes, when carried out entirely manually, are tedious and may further overlook important quality issues due to time and budget pressures. In this paper, we propose *QAssist* – a question-answering (QA) approach that provides automated assistance to stakeholders, including requirements engineers, during the analysis of NL requirements. Posing a question and getting an instant answer is beneficial in various quality-assurance scenarios, e.g., incompleteness detection. Answering requirements-related questions automatically is challenging since the scope of the search for answers can go beyond the given requirements specification. To that end, *QAssist* provides support for mining external domain-knowledge resources. Our work is one of the first initiatives to bring together QA and external domain knowledge for addressing requirements engineering challenges. We evaluate *QAssist* on a dataset covering three application domains and containing a total of 387 question-answer pairs. We experiment with state-of-the-art QA methods, based primarily on recent large-scale language models. In our empirical study, *QAssist* localizes the answer to a question to three passages within the requirements specification and within the external domain-knowledge resource with an average recall of 90.1% and 96.5%, respectively. *QAssist* extracts the actual answer to the posed question with an average accuracy of 84.2%.**

***Index Terms*—Natural-language Requirements, Question Answering (QA), Language Models, Natural Language Processing (NLP), Natural Language Generation (NLG), BERT, T5.**

I. Introduction

A software requirements specification (SRS) is a pivotal artifact in Requirements Engineering (RE). An SRS lays out the desired characteristics, functions, and qualities of a proposed system [1]. SRSs are frequently analyzed by requirements engineers as well as by other stakeholders to ensure the quality of the requirements [2]. To enable the creation of a shared understanding among stakeholders from different backgrounds, e.g., product managers, domain experts, and developers, requirements are most often written in natural language (NL) [3]. Despite its numerous advantages, NL is highly prone to issues such as ambiguity [4], [5], incompleteness [6], [7] and inconsistency [8]. Manually screening for such issues in a lengthy SRS with tens or hundreds of pages is time-consuming, since such screening requires domain knowledge for accurately interpreting the requirements. Evoking domain knowledge is not always quick or easy for humans.

Question answering (QA), i.e., mechanisms to instantly obtain answers to questions posed in NL [9], would be useful as a way to make requirements quality assurance more efficient. To illustrate, consider the example requirements in Fig. 1. These requirements originate from an SRS in the aerospace domain. To facilitate illustration, the requirements in the figure are prefixed with identifiers. For simplicity, we further assume that each requirement in our example is one text passage. In practice, a passage, as we elaborate in Section III, can be made up of multiple consecutive sentences, potentially covering multiple requirements.

While analyzing requirement **DR-13** in Fig. 1, the developer implementing the computations related to the "wet mass" of a spacecraft might come up with question Q1, also shown in Fig. 1. Q1 could be prompted by the developer having doubts about the concept of "wet mass" or them wanting to verify their interpretation. A challenge here is that the answer to a posed question may be absent from the SRS. This happens to be the case for Q1. Since the presence of a requirements glossary cannot be taken for granted either [10], to answer Q1, one may need to consult external domain resources. These resources could be other SRSs from the same domain, or when such SRSs are non-existent or sparse, a domain-specific corpus extracted from a general source such as Wikipedia. On the right side of Fig. 1, we show excerpts of a domain-specific corpus automatically extracted from Wikipedia using an existing open-source corpus extractor [11]. As seen from the figure, just like the SRS being examined, the corpus is made up of passages. These passages may nonetheless be dispersed across multiple documents in the corpus. An answer to Q1 can be found in the extracted corpus. This answer can guide analysts toward making the SRS more complete by providing additional information in the SRS about the concept of "wet mass".

In Fig. 1, we provide two further questions, Q2 and Q3, that can tip analysts to potential quality problems in our example SRS. Automated QA will find **DR-27** and more specifically the highlighted segment in that requirement to be an answer to Q2. Upon examining this answer and not finding the exact frequency of wet-mass computations, the analysts will likely conclude that the SRS is incomplete. For a final example,

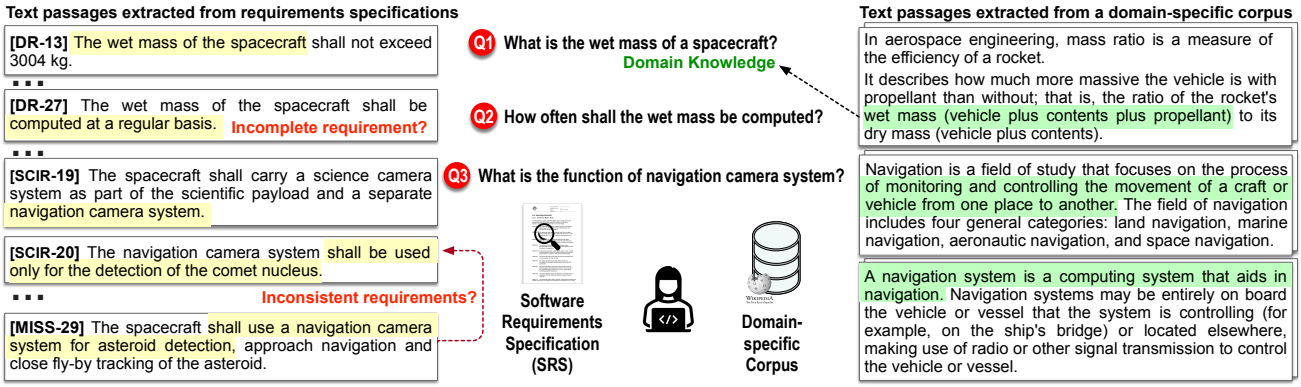

Fig. 1: Example – from left to right: passages from SRS; posed questions; passages from external domain-specific corpus.

consider Q3. In response to this question, QA identifies several likely answers both in the SRS as well as in the extracted corpus. Among the answers are segments from requirements **SCIR-20** and **MISS-29**. Reviewing these two requirements side by side (rather than encountering them potentially many pages apart in the SRS) provides the analysts with a much better chance of noticing the inconsistency between what the two requirements expect of the "navigation camera system". The answers from the domain-specific corpus and the passages where these answers are located provide additional useful information for the review process.

In this paper, we propose *QAssist* – standing for Question Answering Assistance for Improved Requirements Analysis. *QAssist* builds on *open-domain QA*, which is the task of finding in a collection of documents the answer to a given question [12]. *QAssist* takes as input a question posed in NL and returns as output a list of text passages that likely contain the answer to the question. *QAssist* further demarcates a possible answer (text segment) within each retrieved passage. Given questions such as Q1, Q2 and Q3 in Fig. 1, we are interested in two sets of text passages: one obtained from the SRS under analysis (left side of Fig. 1) and the other obtained by mining a domain-specific knowledge resource (right side of Fig. 1). These passages and the answers found within them provide a focused view, helping analysts better understand the requirements and more effectively pinpoint quality problems.

Contributions. Our contributions are as follows:

(1) We devise *QAssist*, an AI-based QA approach aimed at providing assistance with requirements analysis. Given a question posed in NL about the requirements in an SRS, *QAssist* employs Natural Language Processing (NLP) to retrieve two lists of relevant text passages: one from the SRS and one from a domain-specific corpus. In each passage, the likely answer to the posed question is highlighted. When a domain-specific corpus does not exist, *QAssist* automatically builds one, using the phrases appearing in the given SRS as seed terms. Our implementation of *QAssist* is publicly available [13].

(2) We develop in a semi-automatic manner a QA dataset tailored to NL requirements. We name this dataset *REQuestA* – standing for Requirements Engineering Question-Answering dataset. *REQuestA* has been built by two third-party human analysts over six SRSs spanning three application domains. Overall, *REQuestA* contains 387 question-answer pairs. Of these, 214 are manually defined; the remaining 173 are generated automatically and then subjected to manual validation. We make the *REQuestA* dataset publicly available [13].

(3) We empirically evaluate *QAssist* on the *REQuestA* dataset. Our results indicate that *QAssist* retrieves with an accuracy of 100% from a domain-specific corpus the document that is most relevant to a given question. Furthermore, *QAssist* localizes the answer to a question to three passages within the requirements specification and within the corpus with an average recall of 90.1% and 96.5%, respectively. *QAssist* demarcates the actual answer to a question with an average accuracy of 84.2%.

Significance. We believe our work is significant for the RE and NLP communities, as we discuss next. In RE, automated QA has been investigated only to a limited extent and mostly in the context of traceability [14]–[18]. Traceability QA primarily targets the relationship between different artifacts, e.g., requirements, design diagrams and source code. More recently, QA has been studied for improving the understanding of compliance requirements [19]. For RE, the significance of our work is two-fold. First, our QA solution is, to our knowledge, the first to empirically investigate the application of modern QA technologies over industrial requirements. Through a seamless process of posing questions and getting instant answers, our approach enables analysts to explore potential quality issues, e.g., incompleteness and inconsistency. Second, alongside our solution, we build and publicly release a considerably sized QA dataset covering six SRSs from three application domains. This dataset focuses on clarification questions posed over SRSs and is the first dataset of its kind.

QA is widely studied in the NLP community [20], where, as we elaborate in Section VII, many automated solutions and datasets have been proposed and evaluated. The most well-known QA datasets in the NLP literature are derived from Wikipedia, e.g., SQUAD [21], TriviaQA [22] and NQ [23], to name few. There are also examples of domain-specific datasets, e.g., in the medical [24]–[26] and railway [27] do-

mains. From an NLP standpoint, our work is significant in that it is capable of looking beyond a single source for identifying answers to a posed question. The NLP literature concentrates mainly on the situation where the answer to a question resides in an a-priori-known source document (or text passage). Our work departs from this position by bringing in a secondary source of knowledge, namely a domain-specific corpus, to complement the primary source (in our case, an SRS), while maintaining the distinction between the two sources. Using a secondary source is necessitated by our application context: SRSs are typically highly technical with a potentially large amount of tacit (unstated) domain knowledge underpinning them. By provisioning for, and when necessary, automatically constructing a domain-specific corpus, our approach increases the chance that analysts will obtain satisfactory answers to their requirements-related questions.

Structure. Section II presents background. Section III describes our QA approach. Section IV reports on our empirical evaluation. Section V compares with broad-based search engines. Section VI explores threats to validity. Section VII discusses related work. Section VIII concludes the paper.

II. Background

This section describes the background for our QA approach.

Open-domain QA. Our proposed approach targets the open-domain QA task (defined in Section I). Modern open-domain QA solutions work in two stages, combining information retrieval (IR) with machine reading comprehension (MRC) [28]. IR is applied first to narrow the search space by finding the relevant text passages that likely contain the answer to a question [29]. Subsequently, MRC models extract the likely answer to the question from the text passages retrieved [21]. An IR-based method is referred to as a RETRIEVER since it retrieves relevant text, while an MRC-based model is referred to as a READER since it reads the text to find the answer [12]. State-of-the-art QA techniques rely heavily on language models (LMs) such as BERT [30] as an enabling technology [31]. Below, we introduce IR and MRC alongside the LMs that we consider and experiment with in the development of our approach.

Information Retrieval (IR). Given a query and a collection of documents, IR methods are designed to rank the documents according to their relevance to the query [32]. Traditional methods in IR include term frequency - inverse document frequency (TF-IDF) and Okapi Best Matching (BM25). TF-IDF assigns a composite weight to each term occurring in the document depending on its occurrence frequency in the document relative to its frequency in the entire document collection [33]. These weights are used to transform a text sequence into a mathematical vector. Following this, both the query and the documents are represented as vectors, with the query being treated as a (short) document. Relevance is computed using similarity metrics, e.g., cosine similarity [32]. Similarity metrics quantify the similarity between the query and a document while normalizing the difference in vector length; vectors for documents are significantly longer than those for queries. Unlike TF-IDF which is a binary model relying on the presence of question terms in the document collection, BM25 is a probabilistic model that improves the TF-IDF weights using relevance feedback [34].

In the context of QA, IR-based methods assess relevance of documents as well as text passages within individual documents. In the latter case, each passage is regarded as a single document during vectorization. Despite being relatively old, BM25 and to a lesser extent TF-IDF are still widely applied in text retrieval tasks due to their simple implementation and robust behavior [35]. In addition to traditional methods, dense and reranking methods have recently been introduced in the QA literature [35]–[38]. Leveraging language models, dense methods compute relevance based on the text representations in the dense vector space, whereas reranking methods combine the rankings of two different IR-based methods.

Machine Reading Comprehension (MRC). MRC models are specifically used to extract the likely answer to a given question from a text passage [21]. MRC is often solved using pre-trained language models (e.g., BERT), introduced next. These models typically limit the length of the text passage to be less than or equal to 512 tokens [30], [39].

Language Models (LMs). Large-scale neural LMs have rapidly dominated the state-of-the-art in NLP [40]. LMs are pre-trained on large bodies of text in order to learn contextual information, regularities of language, and syntactic and semantic relations between words. This learned knowledge can then be used by fine-tuning LMs to solve downstream NLP tasks [41], e.g., QA [42]. Below, we briefly discuss the LMs that we consider and experiment with in this paper.

Bidirectional Encoder Representations from Transformers (BERT) [30] is pre-trained on the BooksCorpus and English Wikipedia with two training objectives, namely masked language modeling (MLM) and next sentence prediction (NSP). In MLM, a fraction of the tokens in the pre-training text are randomly masked. The model is trained to predict the original vocabulary of these masked tokens based on the surrounding context. For example, BERT should predict the masked token "briefed" in the phrase "`[MASK]` reporters on". In NSP, the model is trained to predict whether two text segments are consecutive in the original text. BERT learns contextualized representations of words by utilizing the *Transformer* architecture [43] and attention mechanisms that allow the model to attend to different information from different representations. For example, the model re-weights the embeddings of "bank" and "river" in the sentence "I walked along the banks of the river" to highlight the meaning of "bank" in this context.

Efficiently Learning an Encoder that Classifies Token Replacements Accurately (*ELECTRA*) [44] improves the contextual representations learned by BERT by replacing the MLM training objective with a token replacement objective, i.e., randomly replacing some tokens instead of masking them.

A Lite BERT (*ALBERT*) [45], A Distilled Version of BERT (*DistilBERT*) [46], *MiniLM* [47] and the Robustly optimized

BERT pre-training approach (*RoBERTa*) [48] are other variants that optimize the size and computational cost of BERT using methods such as knowledge distillation [49] – a technique that transfers knowledge from a large unwieldy model to generate a smaller model with less parameters yet similar performance.

The text-to-text transfer transformer (*T5*) model [50] is another interesting and popular LM. T5 is pre-trained on the Colossal Clean Crawled Corpus (C4) which was also released alongside the model. C4 consists of hundreds of gigabytes of clean text that is crawled from the Web. Compared to BERT-style models, T5 uses a text-to-text framework that enables addressing a wider spectrum of NLP downstream tasks as long as they can be formulated as a text-to-text problem.

III. Approach

In this section, we describe our approach and also establish the notation that we use throughout the rest of the paper. Fig. 2 shows an overview of our approach. *QAssist* takes as input a question (q) posed in NL by the user and an SRS. In step 1, *QAssist* retrieves the most relevant document (d) to q from an external domain-specific corpus ($\mathcal{D}$). In step 2, *QAssist* generates a list of text passages ($\mathcal{T}$) by splitting the input SRS and d. *QAssist* then finds in step 3 the top-k text passages ($\mathcal{R} \subset \mathcal{T}$) that are most relevant to q. In step 4, *QAssist* extracts a likely answer from each text passage retrieved in step 3. *QAssist* finally returns as output the relevant text passages from step 3 alongside the answers extracted in step 4. As explained in Section II, the pipeline for an open-domain QA system like *QAssist* is made up of two phases: (i) IR-based (spanning steps 1 – 3) and (ii) MRC-based (step 4). In phase (i), we apply *two* RETRIEVERS, one for retrieving $d \in \mathcal{D}$ in step 1 (*document retriever* – for short RETRIEVER_D) and another for finding $\mathcal{R}$ in step 3 (*passage retriever* – for short RETRIEVER_T). Next, we elaborate each step of *QAssist*.

A. Step 1: Document Retrieval

As a prerequisite for applying RETRIEVER_D in this step, a corpus $\mathcal{D}$ should be available. When $\mathcal{D}$ is absent, it can be automatically generated using existing corpus-extraction methods [5], [11], [51]–[54]. *QAssist*'s ability to incorporate an external corpus of knowledge into the QA process is important as a way to enrich the output with domain knowledge. In this step, RETRIEVER_D mines $\mathcal{D}$ to find a document d that is most relevant to q. In particular, RETRIEVER_D computes first the relevance between q and each document in $\mathcal{D}$, and then ranks these documents according to relevance scores. From the resulting ranked list, *QAssist* selects as the result of step 1 the most relevant document ($d \in \mathcal{D}$). Note that, while unnecessary for our purposes in this paper, the number of most relevant documents to retrieve from $\mathcal{D}$ can be configured to a value $c > 1$. In that case, the output d from step 1 would be the sequential combination of the top-c retrieved documents.

B. Step 2: Splitting

This step takes two documents as input: the SRS under analysis as well as the most relevant corpus document d retrieved in step 1. *QAssist* automatically generates two lists $\mathcal{T}_S$ and $\mathcal{T}_D$ of text passages by splitting the given SRS and d, respectively. To do so, we employ a simple NLP pipeline that consists of *tokenization* and *sentence splitting*, breaking the input text (SRS and d) into tokens and sentences. Using the annotations from this NLP pipeline, we iterate over each document to identify the text passages.

Recall from Section II that LM-based READERS (which we apply in step 4) typically limit passage length to 512 tokens. Accordingly, we define a *text passage* as a paragraph, unless the paragraph is too long (i.e., has more than 512 tokens) and thus cannot be processed by LMs in its entirety. Long paragraphs are split with one sentence of overlap to preserve context. Concretely, we apply the following procedure to split long paragraphs into coherent passages.

Assume that a given paragraph has a sequence of n sentences, $s_1, \ldots, s_n$. We put consecutive sentences $s_1, \ldots, s_i$ into one passage, such that the length of the resulting passage is less than or equal to 512 tokens. In the next iteration, we start at s_i, i.e., the last sentence of the previous passage. To create the next passage, we take consecutive sentences $s_i, \ldots, s_j$ subject to the 512-token length constraint. This process is repeated until all the sentences in the paragraph have been covered. The rationale for a one-sentence overlap between adjacent passages from the same paragraph is to help maintain flow continuity in the passages.

The output from step 2 ($\mathcal{T}_S$ and $\mathcal{T}_D$) is passed to step 3.

C. Step 3: Passage Retrieval

In this step, we apply RETRIEVER_T to find the k most relevant text passages to q from each $\mathcal{T}_S$ and $\mathcal{T}_D$. We denote the set of resulting passages by $\mathcal{R}_S \subset \mathcal{T}_S$ and $\mathcal{R}_D \subset \mathcal{T}_D$, respectively. In a similar manner to step 1, RETRIEVER_T computes and assigns relevance scores to each text passage in $\mathcal{T}_S$ and $\mathcal{T}_D$. The passages in each $\mathcal{T}_S$ and $\mathcal{T}_D$ are sorted in descending order of relevance and the top-k passages are picked. In Section IV, we empirically assess the implications of the value of k for practice. $\mathcal{R}_S$ and $\mathcal{R}_D$ constitute the input to step 4.

D. Step 4: Answer Extraction

In the last step of *QAssist*, we apply a READER to extract a likely answer to q from each text passage in $\mathcal{R}_S$ and $\mathcal{R}_D$. The likely answers are highlighted in and presented together with $\mathcal{R}_S$ and $\mathcal{R}_D$ as the output of *QAssist*. Which READER technology yields the best results is a question that we investigate empirically in Section IV.

IV. Empirical Evaluation

In this section, we empirically evaluate *QAssist*.

A. Research Questions (RQs)

Our evaluation addresses the following RQs:

RQ1: Which* RETRIEVER *has the highest accuracy in finding text that is most relevant to a given question? Recall from Section III that *QAssist* employs RETRIEVER_D in step 1 (i.e., *Document Retrieval*) and RETRIEVER_T in step 3 (i.e.,

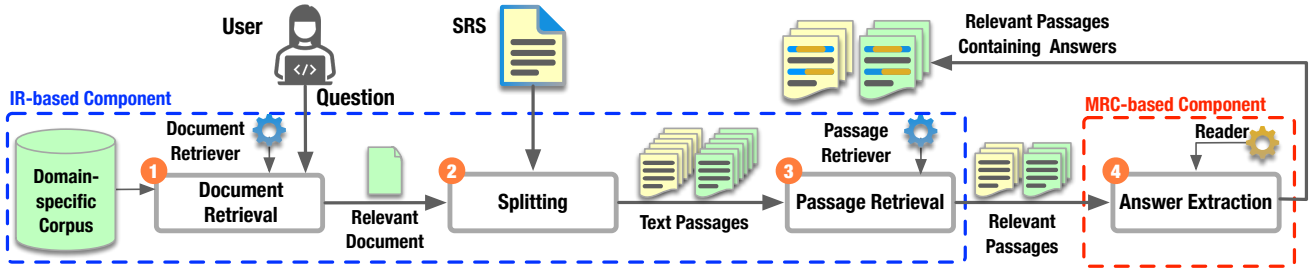

Fig. 2: Overview of our approach (*QAssist*).

Passage Retrieval). RETRIEVER$_D$ takes as input a collection of documents and returns as output the most relevant document $d \in \mathcal{D}$. RETRIEVER$_T$ takes as input a list of text passages and return as output the top-k passages relevant to a given question. For each RETRIEVER, we investigate in RQ1 four alternatives from the IR literature as outlined in Section IV-D. RQ1 identifies the most accurate alternative for each RETRIEVER.

RQ2: Which* READER *produces the most accurate results for extracting the likely answer to a given question? *QAssist* uses in step 4 (i.e., *Answer Extraction*) a READER for extracting a likely answer to a given question from each relevant text passage retrieved by the *passage retrievers* in step 3. Multiple alternative READERS can be applied here as we explain in Section IV-D. RQ2 investigates these alternatives and identifies the most accurate one.

RQ3: Does QAssist run in practical time? RQ3 analyzes *QAssist*'s execution time. To be applicable in practice, *QAssist* needs to be able to answer questions in practical time.

B. Implementation

We implement *QAssist* using Python 3.7.13 and Jupyter Notebooks [55]. Specifically, we implement the NLP pipeline (including the tokenizer and sentence splitter) using the Transformers 3.0.1 library [56]. We implement the traditional IR methods and TF-IDF vectorization using Scikit-learn 1.0.2 [57], and implement BM25 using the BM25 0.2.2 library [58]. The language models that we experiment with include the IR-based models *DistilBERT-base-tas-b* and *MiniLM-L-12-v2* from BeIR [59] and the MRC-based models *ALBERT-large v1.0*, *BERT-large-uncased*, *DistilBERT-base-cased*, *ELECTRA-base*, *MiniLM-uncased* and *RoBERTa-base* from HuggingFace [60]. For corpus extraction from Wikipedia, we use the Wikipedia 1.4.0 library [61]. For question generation, discussed in Section IV-C, we use NLTK 3.2.5 [62] to preprocess text from SRSs and corpus documents. We then apply *T5-base-question-generator* and *BERT-base-cased-qa-evaluator* for automatically generating and assessing question-answer pairs. Both of these models are from HuggingFace.

C. Data Collection Procedure

To evaluate *QAssist*, we collected six SRSs from three application domains, namely aerospace, defence, and security. Our data collection resulted in a QA dataset named *REQuestA* (RE Question-Answering dataset). To reduce the cost and effort required for the construction of this dataset, about half of the question-answer pairs in *REQuestA* were generated automatically using text generation models [50] and then validated by human analysts. The remaining half were defined manually. In this section, we discuss the desiderata for *REQuestA*, the automatic QA generation method, the process for manual definition of question-answer pairs, and finally the details of the resulting dataset.

Desiderata. We identify the following desiderata for *REQuestA* in view of the analytical goals we would like to support, as discussed in Section I.

(1) *Focus on content-based questions*. *REQuestA* is populated with clarification questions over SRSs. *REQuestA* thereby does not contain questions that are not directly related to the SRS content, for instance, questions related to change impact analysis or project management, an example of which would be "How many requirements are not implemented in Phase-1 of the project?". Questions of this nature are legitimate in RE [18], but are outside the scope of our current work.

(2) *Inclusion of external sources of knowledge*. Motivated by covering the domain knowledge that is often left tacit in SRSs, we would like *REQuestA* to include relevant text passages not only from SRSs but also from external sources of knowledge. The inclusion of external knowledge sources enables us to more conclusively evaluate the effectiveness of QA by considering requirements-related questions that would go unanswered based on the contents of a given SRS alone.

QA Auto-generation. Despite the availability of QA datasets, none of them are directly applicable in our work, as explained in Section I. Building a ground truth for QA requires considerable manual effort for proposing both questions and answers. This prompted us to consider question generation (QG) [63], [64] as an aid during dataset construction. QG enables automated derivation of a large number of questions and answers from a given knowledge source; these questions and answers can subsequently be subjected to manual validation for correctness. Such validation generally takes less time and cognitive effort from humans than deriving questions and answers from scratch.

An entry in *REQuestA* is a text passage and a question-answer pair associated with that passage. *An answer* in our work is a short text span in a sentence. The questions and answers in *REQuestA* are derived from two different sources:

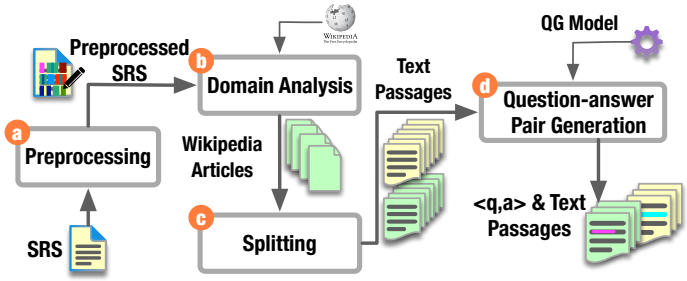

Fig. 3: Overview of our question generation method (used exclusively for building our dataset, *REQuestA*).

the input SRS and a domain-specific corpus created automatically around the content of the input SRS. Fig. 3 shows an overview of our method for automatically generating questions and answers. Given an SRS as input, our method returns a list of question-answer pairs in four steps, elaborated next.

(a) Preprocessing: In this step, we preprocess the input SRS by applying an NLP pipeline. The goal of this step is to identify a set of concepts which are used in the next step to analyze the domain of the input SRS. To find these concepts, we applied REGICE [10] – an existing tool for extracting glossary terms from NL requirements.

(b) Domain Analysis: We build in this step a minimal domain-specific corpus. To do so, we first group the SRSs from the same domain and then use the concepts extracted from these SRSs in step (a). Specifically, we compute for each concept a TF-IDF score, adapted to work over phrases (e.g., "navigation camera") rather than only individual terms (e.g., "camera"). Next, we attempt to increase the specificity of the concepts by removing any generic concepts (e.g., "camera") appearing in WordNet [65] – a generic lexical database for English. We then sort the concepts in descending order of TF-IDF scores and select the top-50 concepts, referring to these concepts as *keywords*. Inspired by recent work on the generation of domain-specific corpora for requirements analysis tasks [5], we use each keyword to query Wikipedia and find a matching article, i.e., an article whose title overlaps with a given keyword. Finally, we randomly select from the matching articles a subset to use in the next step.

(c) Splitting: In this step, we use the same method presented in Section III to automatically split the SRS and Wikipedia articles into a set of text passages.

(d) Question-answer Pair Generation: In this step, we use a QG model based on the T5 language model (introduced in Section II). We give as input a text passage to the QG model. The model first extracts a random answer from the passage and then automatically generates a corresponding question. For example, for passage **DR-13** in Fig. 1, the QG model could first pick "3004 kg", and then generate the following question: *"What shall the wet mass of the spacecraft not exceed?"*. The output of the QG model includes the text passage and a set of automatically generated question-answer pairs. Each such pair will be denoted $\langle q, a \rangle$ hereafter. Note that multiple pairs can be generated from the same text passage. To reduce the manual effort needed for validating the questions and answers, we apply a QA evaluator that is based on BERT. The evaluator takes as input a pair $\langle q, a \rangle$ and returns as output a value representing its prediction about whether the pair is valid. We sort the auto-generated pairs according to the resulting scores from the evaluator, and then select the top 5% of the $\langle q, a \rangle$ pairs automatically generated from each SRS and the Wikipedia articles in the respective corpus.

Construction of REQuestA. The construction of *REQuestA* involved two third-party (non-author) human analysts. The first analyst has a Master's degree in multilingualism. The second analyst has a computer science background with a Master's degree in quality assurance. Both analysts had prior experience with software requirements and had previously contributed to annotation tasks involving SRSs. Before starting their work, the analysts participated in a half-day training session on question answering in RE where they additionally received instructions about the desiderata for *REQuestA*.

We shared with the analysts the original SRSs, the randomly selected Wikipedia articles (created during the domain analysis step in Fig. 3), and the list of automatically generated $\langle q, a \rangle$ pairs for each SRS. The analysts were asked to handle each $\langle q, a \rangle$ pair as follows. Each question q was labeled as *valid* indicating that q was correct as-is, *rephrased* indicating that q was semantically correct but required structural improvement to become valid, or *invalid* indicating that q did not make sense. Similarly, each answer a was labeled as *correct*, *corrected*, or *invalid* with similar indications to the ones mentioned above for q. Additionally, a could be labeled as *not in context* indicating that the question cannot be answered from the given text passage. In this case, we consider the answers as *invalid*. We further asked the analysts to manually define question-answer pairs on each text passage during the validation process. We discuss quality considerations for our dataset later in this section.

To construct the *REQuestA* dataset, we filtered out any pair where either q or a was invalid. For the remaining pairs, we used the rephrased q and corrected a according to the revisions suggested by the human analysts. In total, we automatically generated 204 $\langle q, a \rangle$ pairs; 111 from the SRSs and 93 from the Wikipedia articles. From these, we filtered 31 pairs due to invalid questions or answers, leaving 173 pairs in the dataset (86 from the SRSs and 87 from the Wikipedia articles). We further included in *REQuestA* question-answer pairs that the analysts had defined manually during the validation process alongside the respective text passages. In total, the analysts manually defined 214 pairs (103 from the SRSs and 111 from the Wikipedia articles). Overall, *REQuestA* contains 387 pairs.

Table I provides summary statistics for *REQuestA*. Specifically, the table lists the number of auto-generated $\langle q, a \rangle$ pairs (*auto*) as well as the number of pairs manually defined by the analysts (*man*). The table further shows $\overline{|\mathcal{T}_D|}$ indicating the average number of text passages in the Wikipedia articles (noting that there are multiple articles in each corpus), and

TABLE I: Summary Statistics for the *REQuestA* Dataset.

Domain	$\overline{\lvert\mathcal{T}_D\rvert}$	$\langle q,a\rangle$ *auto*	$\langle q,a\rangle$ *man*	SRS	$\lvert\mathcal{T}_S\rvert$	$\langle q,a\rangle$ *auto*	$\langle q,a\rangle$ *man*
Aerospace	42	45	53	#1	24	8	18
				#2	107	37	40
Defence	94	38	50	#3	11	5	4
				#4	71	19	26
Security	23	4	8	#5	18	15	13
				#6	4	2	2
Total	159	87	111	-	235	86	103

$|\mathcal{T}_S|$ indicating the number of text passages in each SRS.

Quality of REQuestA. As a quality measure, the two analysts reviewed an overlapping subset amounting to 10% of the auto-generated $\langle q, a\rangle$ pairs. We counted an agreement when the analysts selected the same label for a given question or answer (i.e., valid or invalid), noting that valid includes both rephrased and corrected. On this subset, the analysts were in full agreement (i.e., no disagreements) on the labels for the questions and answers.

To further ensure the quality of the dataset, we analyzed all the automatically generated questions and answers against the corrections provided by the human analysts. Out of the 173 valid questions, the analysts collectively rephrased 24 questions (representing ≈14% of the auto-generated questions) and corrected 46 answers (representing ≈26% of the auto-extracted answers). Out of the 46 corrected answers, 26 were expanded by the analysts to include missing tokens, e.g., the auto-extracted answer "software code" was corrected to "implemented software code". To increase the quality of our dataset, we included in *REQuestA* the corrected answers and not the auto-extracted ones. Following best practices in the natural-language generation literature and machine translation [66], we apply BLEU for lexical similarity and BERTScore for semantic similarity. Given two questions, q_1 and q_2, BLEU measures the overlapping tokens between q_1 and q_2. The score is then normalized by the total number of the tokens in q_1 and q_2. BERTScore measures semantic similarity between q_1 and q_2 based on contextual word embeddings. The resulting scores are BLEU=0.54 and BERTScore=0.95. These values indicate that the auto-generated questions and the rephrased ones are semantically very similar albeit using different structures. These scores indicate that our QG method successfully produces semantically correct questions, while also implying that the analysts frequently chose to make structural improvements for better readability.

Since no training or fine-tuning is performed in our approach, we use *REQuestA* in its entirety for empirically evaluating the available QA technologies. To facilitate replication and future research, *REQuestA* is made publicly available [13].

D. Evaluation Procedure

To answer our RQs, we conduct the following experiments. See Section II for background.

EXPI. This experiment answers **RQ1**. We evaluate in EXPI four alternative RETRIEVERS, including the traditional RETRIEVERS TF-IDF and BM25, DistilBERT dense RETRIEVER, and a reranking RETRIEVER that pairs BM25 with MiniLM cross encoder. We identify in EXPI the most accurate RETRIEVER applied in step 1 of our approach (Fig. 2) for retrieving the most relevant external document from a domain-specific corpus. We further identify the most accurate RETRIEVER in step 3 for retrieving from the input SRS and the most relevant external document the top-k relevant text passages for a given question. We compare the performance of the alternative RETRIEVERS using two evaluation metrics commonly used in the IR literature [29]. The first metric is *recall@k (R@k)* and assesses whether the document (or text passage) containing the correct answer to a given question (q) is in the ranked list of the top-k documents (or passages) produced by the RETRIEVER. The second metric, *normalized discounted cumulative gain@k (nDCG@k)*, is similar to *R@k*, except that it accounts not only for the mere presence of the relevant document (or passage) but also for its rank.

We note that we are interested only in the most relevant document (top-1) retrieved by the document RETRIEVER in step 1 of our approach. In this case, ranking is not relevant and the above two metrics produce the same result; we thus report only R@1 for the document RETRIEVER. To run EXPI, using an existing open-source tool [11], we generate domain-specific corpora covering the aerospace, defence, and security domains and corresponding to the SRSs in our study.

EXPII. This experiment answers **RQ2**. To extract the answer to a given question in step 4 of our approach (Fig. 2), we experiment with the following alternative READERS: ALBERT, BERT, DistilBERT, ELECTRA, MiniLM, and RoBERTa. We compare the performance of the READERS using *Accuracy (A)*, computed as the number of questions correctly answered by the READER divided by the total number of questions. To decide whether an answer is correct, we compare the extracted answer by the READERS against the answer provided by the analysts in our dataset (*REQuestA*). We evaluate an extracted answer for correctness in three different modes. Let a_{GT} denote the ground-truth answer to a question. In *exact matching* mode, the extracted answer fully matches a_{GT}. In *partial matching* mode, the extracted answer partially matches (i.e., overlaps with) a_{GT}. In *semantic matching* mode, the extracted answer has a cosine semantic similarity with a_{GT} that is greater than a predefined threshold. In our work, we apply a threshold of 0.5 [67]. The first two modes evaluate correctness at a lexical level, whereas the last mode measures correctness based on meaning.

In addition to reporting accuracy, we also report F1 measure – another commonly-reported lexical metric in the QA literature [68]. F1 is the harmonic mean computed as $2 * P * R/(P + R)$, where P is the precision and R is the recall. We define P as the number of overlapping tokens between the extracted answer and a_{GT} divided by the total number of tokens in the extracted answer. We define R as the

TABLE II: R@1 of *Document* RETRIEVER (**RQ1**).

Domain	$\lvert\mathcal{D}\rvert^{\dagger}$	TF-IDF	BM25	Dense	Reranking
Aerospace	1158	**100**	**100**	99.0	**100**
Defence	781	**100**	**100**	98.9	**100**
Security	50	**100**	**100**	91.7	**100**

† $|\mathcal{D}|$ is the number of articles in the corpus ($\mathcal{D}$) of Wikipedia articles.

number of overlapping tokens between the extracted answer and a_{GT} divided by the total number of tokens in a_{GT}. We report in EXPII overall F1-score averages for all questions.

EXPIII. This experiment answers **RQ3**. We report the execution of our approach with the most accurate models from the previous experiments. EXPIII is conducted on the Google Colaboratory cloud using the free plan with the following specifications: Intel(R) Xeon(R) CPU@2.20GHz, Tesla T4 GPU, and 13GB RAM.

E. Answers to the RQs

RQ1. Which* RETRIEVER *has the highest accuracy in finding text that is most relevant to a given question? RQ1 identifies the best-performing (i) *document* RETRIEVER and (ii) *passage* RETRIEVER to be applied in steps 1 and 3 of *QAssist*, respectively. Tables II and III show the results of EXPI.

In Table II, traditional RETRIEVERS (TF-IDF and BM25) are clearly able to find the most relevant documents across all domains, thus achieving a perfect R@1. In comparison, our dense RETRIEVER (DistilBERT) has an average R@1 of 96.5%, which is slightly worse than the traditional RETRIEVERS. The reranking RETRIEVER achieves a perfect R@1 as well since it partially uses the results of BM25. In view of these results, we select BM25 as the RETRIEVER to use for step 1 of our approach, since BM25 is computationally more efficient than the reranking RETRIEVER. Compared to TF-IDF, BM25 is more robust [69] and widely-applied in the QA literature [35].

In Table III, we show the results for retrieving the most relevant k text passages for $k = 1, 3, 5, 10$. The upper part of the table provides the average results for our collection of six SRSs. The lower part of the table shows the results for retrieving passages from the most relevant external document. We recall from Section III that $\mathcal{T}_S$ denotes the set of passages within a given SRS and $\mathcal{T}_D$ denotes the passages in the most relevant external document from the corpus. In our dataset, an SRS has on average about 40 passages, whereas an external document has on average 53 passages. Here, recall measures the presence of the relevant passage in the retrieved passages, whereas nDCG measures whether the relevant passage has a higher rank among the retrieved passages. In our analysis, we focus on recall, noting that rank does not play as significant a role for small values of k (≤ 3) where our discussion of recall, below, leads us to.

We observe from Table III that the reranking RETRIEVER outperforms the alternatives in the two metrics and for all k values, except for the security domain as we elaborate later. We naturally see improvement in recall with higher values of k.

TABLE III: Accuracy of *Passage* RETRIEVER (**RQ1**).

		Top-1		Top-3		Top-5		Top-10	
From $\mathcal{T}_S$		R	nDCG	R	nDCG	R	nDCG	R	nDCG
	(1)	60.3	60.3	78.6	70.9	84.3	73.3	89.8	75.1
	(2)	62.8	62.8	78.5	71.9	85.4	74.6	**92.4**	76.9
	(3)	52.9	52.9	81.2	70.3	86.5	72.5	88.5	73.2
	(4)	**78.9**	**78.9**	**90.1**	**85.8**	**92.2**	**86.6**	**92.4**	**86.7**
From $\mathcal{T}_D$		R	nDCG	R	nDCG	R	nDCG	R	nDCG
Aerospace	(1)	43.6	43.6	64.7	56.0	68.6	57.5	91.5	**94.9**
	(2)	50.5	50.5	83.0	70.1	91.7	73.6	**95.0**	74.7
	(3)	66.7	66.7	87.3	79.5	90.6	80.8	91.3	81.0
	(4)	**75.1**	**75.1**	**95.0**	**87.3**	**95.0**	**87.3**	**95.0**	87.3
Defence	(1)	41.9	41.9	62.1	54.1	66.7	55.9	86.3	62.2
	(2)	38.0	38.0	81.2	64.1	89.3	67.3	**94.6**	69.1
	(3)	**77.2**	**77.2**	89.8	84.7	**91.2**	85.2	92.5	85.6
	(4)	76.0	76.0	**94.6**	**87.6**	94.6	**87.6**	**94.6**	**87.6**
Security	(1)	33.4	33.4	70.0	53.8	70.0	53.8	80.0	57.4
	(2)	43.3	43.3	70.0	59.3	**100**	71.3	**100**	71.3
	(3)	63.3	63.3	**100**	85.2	**100**	85.2	**100**	85.2
	(4)	**80.0**	**80.0**	**100**	**92.6**	**100**	**92.6**	**100**	**92.6**

(1) TF-IDF, (2) BM25, (3) Dense, and (4) Reranking.

Concretely, the reranking RETRIEVER achieves for retrieving passages from the SRSs an average recall of 78.9%, 90.1%, 92.2%, and 92.4% at $k = 1$, $k = 3$, $k = 5$, and $k = 10$, respectively. The same RETRIEVER achieves for retrieving passages from the external document an average recall of 77.0% at $k = 1$, and 96.5% at $k = 3$, $k = 5$, and $k = 10$.

Selecting the best value of k has practical implications. While higher k values yield better recall, they entail additional effort for reviewing the results of *QAssist*. For instance, selecting $k = 10$ yields the best overall results, which implies that a stakeholder has more relevant context at their disposal for understanding and interpreting the requirements. However, this comes at the cost of more time and effort needed to browse through the retrieved text passages. We deem $k = 3$ as a reasonable compromise in our context, since the gain in recall at $k = 5$ (in comparison to $k = 3$) is merely $\approx$2 percentage points; selecting $k = 5$ would imply browsing through two additional passages per question. That said, k can be left as a user-configurable parameter, to be adjusted according to needs and the time budget available.

The results show that the dense RETRIEVER, DistilBERT, performs on par with the reranking RETRIEVER for the security domain. In our collection, the domain-specific corpus generated for security is the smallest among the corpora as it is generated from two SRSs, one of which is very small (SRS #6). Furthermore, the number of passages analyzed in this domain is 23, compared to the aerospace and defence with an average of 42 and 94 passages, respectively. This observation suggests that the dense RETRIEVER is more effective when there is a fewer number of passages. The performance of the reranking RETRIEVER is in general better than that of the dense RETRIEVER for $k = 3$. Consequently, we select the reranking RETRIEVER as the best-performing alternative for step 3 of our approach.

The answer to **RQ1** is that BM25 is the best document RETRIEVER with a perfect recall, and the reranking RETRIEVER is the best passage RETRIEVER with an average recall@3 of 90.1% and 96.5% for SRSs and external (corpus) documents, respectively.

RQ2. Which* READER *produces the most accurate results for extracting the likely answer to a given question? Table IV shows the results of EXPII, comparing the accuracy of the READERS for extracting the answer to a given question. Note that in RQ1, we focused on retrieving *passages*, whereas in RQ2, we are interested in determining which READER identifies the most accurate *text span* containing the answer within the passages already found.

The table shows that the most accurate READER varies depending on which matching mode we choose. Considering the *exact matching* mode, RoBERTa is the most accurate READER, followed by ALBERT, with an average accuracy of 24.6% and 24.3%, respectively. This finding is corroborated by the F1 measure. Nevertheless, both READERS are outperformed by DistilBERT in the *partial matching* mode which achieves the best average accuracy of 86.4%.

Noting their lexical nature, the exact and partial matching modes as well as the F1 measure have the drawback that they focus on whether the extracted answer is literally the same as the one in the ground truth rather than providing equivalent information [70]. For example, consider question Q1 in Fig. 1. The answer extracted for this question from the first passage of the domain-specific corpus (right side of the figure) could be the following: "how much more massive the vehicle is with propellant than without". This answer does not have a lexical overlap with the highlighted answer (shaded green in the figure), despite considerable similarity in meaning. For such cases, lexical metrics would evaluate the extracted answer as incorrect. To better assess the performance of the READERS in our context, where users may be seeking all closely relevant information, we further report results for the *semantic matching* mode. Using the *semantic matching* mode would lead us to the same conclusion as that offered by *exact matching* and F1. That is, ALBERT and RoBERTa have the highest average accuracy of 84.2% and 84.0%, respectively. Despite the similar behavior of the two models, ALBERT considerably outperforms RoBERTa in *partial matching* mode with an average percentage points of ≈19%. We thus select ALBERT as the best-performing READER for answer extraction.

Since Wikipedia has been used for pre-training BERT and many variants thereof, and considering that part of our question-answer pairs originate from Wikipedia, we show that answer extraction in our approach is still accurate for content that originates from sources different from Wikipedia. Recall from Table I that REQuestA contains a total of 189 (= 86 + 103) question-answer pairs from SRSs and another 198 (= 87 + 111) pairs from Wikipedia articles. The 189 question-answer pairs from the SRSs are independent from Wikipedia. The performance of BERT-based models over these pairs is a representative indicator for non-Wikipedia content.

TABLE IV: READER Accuracy Results (**RQ2**); table further shows loading time for READERS (a consideration for **RQ3**).

Model	Accuracy			F1	Time
	Exact	*Partial*	*Semantic*		
ALBERT	24.3	79.1	**84.2**	64.6	193.2
q_S	31.7	78.0	86.4	67.6	
q_D	17.2	80.2	82.1	61.7	
BERT	21.4	70.6	82.9	63.1	32.2
q_S	28.3	70.4	83.8	66.4	
q_D	14.8	70.8	82.1	59.9	
DistilBERT	23.0	**86.4**	75.9	61.0	5.8
q_S	32.7	86.4	77.3	67.5	
q_D	13.7	86.5	74.6	54.8	
ELECTRA	21.1	81.3	81.0	60.1	19.1
q_S	31.2	82.1	80.6	65.0	
q_D	11.5	80.5	81.4	55.4	
MiniLM	23.3	73.3	82.4	63.4	5.0
q_S	32.4	73.6	82.6	66.1	
q_D	14.6	73.0	82.2	60.9	
RoBERTa	**24.6**	60.2	84.0	**65.2**	11.6
q_S	32.8	61.0	84.3	70.1	
q_D	16.8	59.4	83.7	60.5	

The table reports performance results for all question-answer pairs as well as for SRS-based (q_S) and domain-based (q_D) pairs separately.

In Table IV, we further provide a breakdown of the READER results based on the origin of the question-answer pairs. We denote SRS-based questions as q_S and domain-based questions (which, in our case study, are sourced from Wikipedia) as q_D. The table shows that all models achieve on-par or better accuracy over q_S compared to q_D. Based on the breakdown in Table IV, we conclude that the exposure of BERT-based models to Wikipedia during pre-training is unlikely to have influenced our performance results.

The answer to **RQ2** is that considering both lexical and semantic measures, ALBERT provides the best overall trade-off for answer extraction with an average accuracy of ≈24% in the *exact matching* mode, ≈79% in the *partial matching* mode, and ≈84% in the *semantic matching* mode.

RQ3. Does QAssist run in practical time? To answer RQ3, we discuss the execution time of our approach based on the conclusions from RQ1 and RQ2 and the setup described under *EXPIII* in Section IV-D. Based on RQ1, we select BM25 as the document RETRIEVER and the reranking method as the passage RETRIEVER. For answer extraction, based on RQ2, we select ALBERT as the READER. With these choices, we report the execution time for each step of *QAssist* (Fig. 2).

Retrieving the most relevant document from the corpora created for the aerospace, defence, and security domains (step 1) requires 2.06, 1.37, and 0.08 seconds, respectively. The time required in step 2 for splitting a document into tokens and sentences is comparatively negligible. For retrieving relevant

passages in step 3, we note that the six SRSs in our study vary in size from small (SRS#6 with 32 requirements) to large (SRS#2 with 1041 requirements). Similarly, the Wikipedia articles (making up the domain-specific corpora) from which we retrieve passages vary in size, as shown previously in Table I. For our dataset, the time required for retrieving passages from an SRS is 2.27 seconds for the smallest SRS and 6.43 seconds for the largest. For corpus articles, the average time for passage retrieval is 2.62 seconds. Extracting answers from passages, i.e., step 4, takes an average of 1.1 seconds.

In addition to the above-reported execution times, there is a one-time loading overhead for the READER, as shown in the last column of Table IV. For ALBERT (best READER from RQ2), this overhead is ≈3.2 minutes. We deem this overhead acceptable considering that, once the READER has been loaded, the user can ask as many questions as desired.

Excluding the overhead for loading the READER, answering an individual question, when averaged across all questions in our dataset, takes 10.36 seconds. We believe this execution time is reasonable for most practical situations. Moreover, the execution time can be improved if one has access to more powerful computing resources than ours (Google Colab's free plan, as noted in Section IV-D).

> When run on Google Colab's free plan, our approach takes an average of 10.36 seconds to answer an individual question. In addition, one has to provision for a one-time overhead of 3.2 minutes to load the required language model (ALBERT). We find this level of performance practical for question answering over requirements. Performance can be further improved with more powerful computational resources for language models.

V. Comparison with Broad-based Search Engines

An intuitive way for QA during the analysis of an SRS would be to pose the questions to a (broad-based) search engine such as Google. In the context of our work, search engines are generally not very effective for two main reasons. First, answers to domain-specific questions can reside in company-specific documents which are unlikely to be accessible to search engines. Our approach, in contrast, gives analysts the possibility to plug company-specific documents into the QA system. Second, the lack of domain-specificity in search engines can easily result in misleading answers. For example, an online search for "rocket mass" instead of "wet mass" to answer Q1 in Fig. 1 would point the analyst to the design of a rocket mass heater[1], which is not relevant to the space domain. Unlike search engines, our approach is scoped to the original SRS and any external knowledge resources selected by the user. As such, questions are implicitly disambiguated as long as the external knowledge resources are domain-specific. To further illustrate, consider the question "What is NEAT?". Posing this question online would lead to irrelevant results due to the ambiguous abbreviation, whereas posing the same question to our approach would retrieve the definition of "Near-Earth Asteroid Tracking" – inline with the SRS content.

To empirically assess the success rate of search engines in our problem context, we posed to Google from our dataset a total of 50 verbatim questions. Of these, 20 questions were SRS-based and 30 were domain-based. The authors independently investigated whether the top-3 retrieved documents by Google contained the correct answer as per our ground truth. Out of the 50 questions, we found that 16 questions were answered correctly by Google, leading to a success rate of 32%. From the 16 correctly answered questions, 14 were domain-based. We note that the domain-based questions in our dataset, REQuestA, originate from Wikipedia articles, which search engines have access to and can crawl. The outcome would most likely have been different had the external knowledge resource not been public. Therefore, in addition to the need for explicit disambiguation as discussed above, the success rate of search engines is likely to be affected by the public accessibility of the documents that should be considered during QA. In conclusion, we believe that search engines are currently not the best alternative for QA over specialized and proprietary material – a situation that is common in RE.

VI. Threats to Validity

The validity concerns most pertinent to our evaluation are internal and external validity.

Internal Validity. The main concern regarding internal validity is dataset bias. To mitigate bias, the authors ensured that they were not involved in dataset construction; this task was done exclusively by third parties (non-authors) who had no exposure to our technical solution.

External Validity. Our evaluation is based on a dataset containing six industrial SRSs and spanning three different application domains. The results we obtained across these SRSs and domains combined with the comparatively large size of our QA dataset provide confidence about the generalizability of our empirical findings. Additional experimentation is nevertheless important to further mitigate external-validity threats.

VII. Related Work

In this section, we position our work in the existing literature on QA as studied by the RE and NLP communities.

QA in RE. There has been only limited research where QA is applied for addressing RE problems. Existing works focus on requirements traceability [15]–[17], identifying compliance requirements [19], [71], and extracting information from online forums [72]. These techniques are mostly IR-based, with the exception of [19], which, like our approach, uses machine reading comprehension (MRC). Our approach differs from [19] both in its purpose and also in how it employs MRC. First, whereas [19] focuses on QA over legal provisions (e.g., privacy regulations), our approach deals with QA over SRSs. Second, [19] is limited in that it applies MRC to a-priori-specified documents only. Our approach can, in contrast, mine domain-related content from Wikipedia in an attempt to make

[1] https://en.wikipedia.org/wiki/Rocket_mass_heater

tacit domain knowledge explicit and thereby handle questions that would go unanswered if the scope of search for answers was limited to the SRS under analysis only.

In terms of QA datasets, not many such datasets are available in RE. Abualhaija et al.'s dataset of 107 question-and-answer pairs [19] is built over legal documents. In contrast, our dataset, *REQuestA*, is built over SRSs. To our knowledge, *REQestA* is the first dataset of its kind, providing a total of 387 question-and-answer pairs on industrial requirements.

Malviya et al. [18] investigate questions that requirements engineers typically ask throughout the development process. They collect through a survey with industry practitioners a set of 159 questions, grouped into nine different categories such as project management and quality assessment. Malviya et al.'s questions are broad and can crosscut several artifacts in the development life cycle. Our work focuses specifically on clarification questions asked over SRSs and associated domain-knowledge resources; our objective here is developing automated QA technologies that can answer such questions.

QA in NLP. QA tasks in the NLP literature include question classification, answer extraction, and question generation [73]–[76]. Answer extraction is considered to be the main QA task in NLP [77]. Recent advances in QA answer extraction include fine-tuning large scale language models such as BERT, RoBERTa, and ALBERT [78]–[81]. Inspired by the NLP literature, we apply in our work the QA models reported in a recent QA benchmark [35]. Several existing QA datasets curated from generic text are publicly available. These datasets include SQuAD [21], GLUE [82], and TriviaQA [22]. There are also some domain-specific datasets, e.g., for the medical [25] and railway [27] domains. For the same reasons mentioned earlier when discussing related work in RE, none of the available datasets in NLP are suitable for our needs in this paper.

Language models have been employed for various text generation tasks [64], including question generation (QG) [50], [83]. QG models have enabled researchers in many fields to automatically generate their own synthetic QA datasets [84]–[87]. Our dataset was partially generated using QG. To our knowledge, QG has not been attempted in RE before.

Our work is distinguished from QA research in NLP in that we provide an end-to-end solution. Our approach covers all QA steps starting from posing a question down to providing the most relevant passages and potential answers. Foundational research in NLP often focuses on individual QA steps, e.g., IR-based text retrieval or MRC-based answer extraction. Our work does not contribute to the foundations for QA. Nevertheless, our motivating use case (QA over requirements), our combination of NLP technologies, the flexibility to build domain-specific corpora and consult them during QA, and our extensive empirical evaluation of QA in an RE context are, to the best of our knowledge, new.

VIII. Conclusion

In this paper, we proposed *QAssist* – an AI-based question-answering (QA) system to support the analysis of natural-language requirements. Given a question, *QAssist* retrieves relevant text passages from both the requirements document being analyzed as well as an external source of domain knowledge. *QAssist* further highlights the likely answer to the question in each retrieved text passage. The flexibility to incorporate an external knowledge source into the QA process enables *QAssist* to answer otherwise unanswerable questions related to the tacit domain information assumed by the requirements. When a domain-knowledge resource is absent, *QAssist* automatically builds one by mining Wikipedia articles, using the terminology in the requirements being analyzed to guide the mining process. To evaluate *QAssist*, we created through third-party annotators a QA dataset, named *REQuestA*. Both *QAssist* and *REQuestA* are publicly available [13]. Our empirical results indicate that *QAssist* localizes the answer to a posed question to three passages within the requirements document and within the external domain-knowledge resource with an average recall of 90.1% and 96.5%, respectively. Narrowing the scope to these passages, *QAssist* has an average accuracy of 84.2% in pinpointing the actual answer.

In future work, we would like to conduct user studies to better understand how practitioners would interact with requirements documents when equipped with a QA tool. Another future direction is to experiment with emerging QA methods in NLP that are capable of producing a "no answer" outcome when a question is not answerable with sufficient accuracy.

Acknowledgements. This work was funded by Luxembourg's National Research Fund (FNR) under the grant BRIDGES18/IS/12632261 and NSERC of Canada under the Discovery and Discovery Accelerator programs. We are grateful to the research and development team at QRA Corp. for valuable insights and assistance.

References

[1] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, 1st ed. Wiley, 2009.

[2] K. Pohl, *Requirements Engineering*, 1st ed. Springer, 2010.

[3] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing (nlp) for requirements engineering: A systematic mapping study," *arXiv preprint arXiv:2004.01099*, 2020.

[4] A. Ferrari and A. Esuli, "An NLP approach for cross-domain ambiguity detection in requirements engineering," *Automated Software Engineering*, vol. 26, no. 3, 2019.

[5] S. Ezzini, S. Abualhaija, C. Arora, M. Sabetzadeh, and L. C. Briand, "Using domain-specific corpora for improved handling of ambiguity in requirements," in *2021 IEEE/ACM 43rd International Conference on Software Engineering*, 2021.

[6] F. Dalpiaz, I. Schalk, and G. Lucassen, "Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP," in *Proceedings of the 24th Working Conference on Requirements Engineering: Foundation for Software Quality*, 2018.

[7] C. Arora, M. Sabetzadeh, and L. C. Briand, "An empirical study on the potential usefulness of domain models for completeness checking of requirements," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2509–2539, 2019.

[8] I. Hadar, A. Zamansky, and D. M. Berry, "The inconsistency between theory and practice in managing inconsistency in requirements engineering," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3972–4005, 2019.

[9] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed., 2020, https://web.stanford.edu/~jurafsky/slp3/(visited 2021-06-04).

[10] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated extraction and clustering of requirements glossary terms," *IEEE Transactions on Software Engineering*, vol. 43, no. 10, 2017.

[11] S. Ezzini, S. Abualhaija, and M. Sabetzadeh, "Wikidominer: Wikipedia domain-specific miner," in *Proceedings of the 17th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2022.

[12] F. Zhu, W. Lei, C. Wang, J. Zheng, S. Poria, and T.-S. Chua, "Retrieving and reading: A comprehensive survey on open-domain question answering," *arXiv preprint arXiv:2101.00774*, 2021.

[13] "Replication package," 2022. [Online]. Available: https://gitlab.uni.lu/sezzini/QAssist/

[14] J. I. Maletic and M. L. Collard, "Tql: A query language to support traceability," in *2009 ICSE workshop on traceability in emerging forms of software engineering*. IEEE, 2009, pp. 16–20.

[15] P. Mäder and J. Cleland-Huang, "A visual language for modeling and executing traceability queries," *Software & Systems Modeling*, vol. 12, no. 3, pp. 537–553, 2013.

[16] P. Pruski, S. Lohar, W. Goss, A. Rasin, and J. Cleland-Huang, "Tiqi: answering unstructured natural language trace queries," *Requirements Engineering*, vol. 20, no. 3, pp. 215–232, 2015.

[17] J. Lin, Y. Liu, J. Guo, J. Cleland-Huang, W. Goss, W. Liu, S. Lohar, N. Monaikul, and A. Rasin, "Tiqi: A natural language interface for querying software project data," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2017, pp. 973–977.

[18] S. Malviya, M. Vierhauser, J. Cleland-Huang, and S. Ghaisas, "What questions do requirements engineers ask?" in *2017 IEEE 25th International Requirements Engineering Conference*. IEEE, 2017, pp. 100–109.

[19] S. Abualhaija, C. Arora, A. Sleimi, and L. Briand, "Automated question answering for improved understanding of compliance requirements: A multi-document study," in *In Proceedings of the 30th IEEE International Requirements Engineering Conference, Melbourne, Australia 15-19 August 2022*, 2022.

[20] M. A. C. Soares and F. S. Parreiras, "A literature review on question answering techniques, paradigms and systems," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 6, pp. 635–646, 2020.

[21] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[22] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension," *arXiv preprint arXiv:1705.03551*, 2017.

[23] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee *et al.*, "Natural questions: a benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019.

[24] A. Pampari, P. Raghavan, J. Liang, and J. Peng, "emrqa: A large corpus for question answering on electronic medical records," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 2357–2368.

[25] J. He, M. Fu, and M. Tu, "Applying deep matching networks to chinese medical question answering: a study and a dataset," *BMC medical informatics and decision making*, vol. 19, no. 2, pp. 91–100, 2019.

[26] Y. Tian, W. Ma, F. Xia, and Y. Song, "Chimed: A chinese medical corpus for question answering," in *Proceedings of the 18th BioNLP Workshop and Shared Task*, 2019, pp. 250–260.

[27] Z. Hu, "Research and implementation of railway technical specification question answering system based on deep learning," in *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 2020, pp. 5–9.

[28] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading Wikipedia to answer open-domain questions," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2017, pp. 1870–1879.

[29] M. McGill and G. Salton, *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018.

[31] S. Liu, X. Zhang, S. Zhang, H. Wang, and W. Zhang, "Neural machine reading comprehension: Methods and trends," *Applied Sciences*, vol. 9, no. 18, p. 3698, 2019.

[32] C. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*, 1st ed. Cambridge University Press, 2008.

[33] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.

[34] S. Robertson and H. Zaragoza, *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[35] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych, "Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models," *arXiv preprint arXiv:2104.08663*, 2021.

[36] R. Nogueira and K. Cho, "Passage re-ranking with bert," *arXiv preprint arXiv:1901.04085*, 2019.

[37] K. Wang, N. Thakur, N. Reimers, and I. Gurevych, "Gpl: Generative pseudo labeling for unsupervised domain adaptation of dense retrieval," *arXiv preprint arXiv:2112.07577*, 2021.

[38] S. Zhuang and G. Zuccon, "Dealing with typos for bert-based passage retrieval and ranking," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 2836–2842.

[39] D. Chen and W.-t. Yih, "Open-domain question answering," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*. Online: Association for Computational Linguistics, 2020, pp. 34–37.

[40] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[41] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[42] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?" *arXiv preprint arXiv:1909.01066*, 2019.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[44] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[45] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[46] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[47] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5776–5788, 2020.

[48] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[49] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[50] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," 2019.

[51] D. Milne, O. Medelyan, and I. Witten, "Mining domain-specific thesauri from wikipedia: A case study," in *Proceedings of the 5th IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, 2006.

[52] G. Cui, Q. Lu, W. Li, and Y. Chen, "Corpus exploitation from Wikipedia for ontology construction," in *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*. Marrakech, Morocco: European Language Resources Association (ELRA), May 2008.

[53] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "Pure: A dataset of public requirements documents," in *2017 IEEE 25th International Requirements Engineering Conference*, 2017.

[54] K. Saxena, T. Singh, A. Patil, S. Sunkle, and V. Kulkarni, "Leveraging Wikipedia navigational templates for curating domain-specific fuzzy conceptual bases," in *Proceedings of the Second Workshop on Data Science with Human in the Loop: Language Advances*. Online: Association for Computational Linguistics, Jun. 2021, pp. 1–7.

[55] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016.

[56] 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. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 2020.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[58] B. Dorian, J. Sarthak, N. Vít, and nlp4whp, "dorianbrown/rank_bm25," 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6106156

[59] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych, "BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[60] "Hugging face," 2022. [Online]. Available: https://huggingface.co/

[61] J. Goldsmith, "The wikipedia libray," 2022. [Online]. Available: https://pypi.org/project/wikipedia/

[62] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, 2002.

[63] X. Du and C. Cardie, "Identifying where to focus in reading comprehension for neural question generation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, pp. 2067–2073.

[64] L. Pan, W. Lei, T.-S. Chua, and M.-Y. Kan, "Recent advances in neural question generation," *arXiv preprint arXiv:1905.08949*, 2019.

[65] G. Miller, "WordNet: A lexical database for English," *Communications of the ACM*, vol. 38, no. 11, 1995.

[66] M. Hanna and O. Bojar, "A fine-grained analysis of BERTScore," in *Proceedings of the Sixth Conference on Machine Translation*. Online: Association for Computational Linguistics, 2021, pp. 507–517.

[67] D. Ramage, A. N. Rafferty, and C. D. Manning, "Random walks for text semantic similarity," in *Proceedings of the 2009 workshop on graph-based methods for natural language processing (TextGraphs-4)*, 2009, pp. 23–31.

[68] B. B. Cambazoglu, M. Sanderson, F. Scholer, and B. Croft, "A review of public datasets in question answering research," in *ACM SIGIR Forum*, vol. 54, no. 2. ACM New York, NY, USA, 2021, pp. 1–23.

[69] J. S. Whissell and C. L. Clarke, "Improving document clustering using okapi bm25 feature weighting," *Information retrieval*, vol. 14, no. 5, pp. 466–487, 2011.

[70] J. Risch, T. Möller, J. Gutsch, and M. Pietsch, "Semantic answer similarity for evaluating question answering models," *arXiv preprint arXiv:2108.06130*, 2021.

[71] A. Sleimi, M. Ceci, N. Sannier, M. Sabetzadeh, L. Briand, and J. Dann, "A query system for extracting requirements-related information from legal texts," in *27th IEEE International Requirements Engineering Conference*. IEEE, 2019.

[72] G. M. Kanchev, P. K. Murukannaiah, A. K. Chopra, and P. Sawyer, "Canary: an interactive and query-based approach to extract requirements from online forums," in *2017 IEEE 25th International Requirements Engineering Conference*. IEEE, 2017, pp. 470–471.

[73] T. Hao, X. Li, Y. He, F. L. Wang, and Y. Qu, "Recent progress in leveraging deep learning methods for question answering," *Neural Computing and Applications*, pp. 1–19, 2022.

[74] A. A. Yusuf, F. Chong, and M. Xianling, "An analysis of graph convolutional networks and recent datasets for visual question answering," *Artificial Intelligence Review*, pp. 1–24, 2022.

[75] H. Jin, Y. Luo, C. Gao, X. Tang, and P. Yuan, "Comqa: Question answering over knowledge base via semantic matching," *IEEE Access*, vol. 7, pp. 75 235–75 246, 2019.

[76] D. Diefenbach, A. Both, K. Singh, and P. Maret, "Towards a question answering system over the semantic web," *Semantic Web*, vol. 11, no. 3, pp. 421–439, 2020.

[77] B. Ojokoh and E. Adebisi, "A review of question answering systems," *Journal of Web Engineering*, vol. 17, no. 8, pp. 717–758, 2018.

[78] L. Jing, C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljacic, and Y. Bengio, "Gated orthogonal recurrent units: On learning to forget," *Neural computation*, vol. 31, no. 4, pp. 765–783, 2019.

[79] A. Wulamu, Z. Sun, Y. Xie, C. Xu, and A. Yang, "An improved end-to-end memory network for qa tasks," *CMC-COMPUTERS MATERIALS & CONTINUA*, vol. 60, no. 3, pp. 1283–1295, 2019.

[80] Q. Ren, X. Cheng, and S. Su, "Multi-task learning with generative adversarial training for multi-passage machine reading comprehension," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8705–8712.

[81] T. Parshakova, F. Rameau, A. Serdega, I. S. Kweon, and D.-S. Kim, "Latent question interpretation through variational adaptation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 11, pp. 1713–1724, 2019.

[82] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[83] V. Kumar, Y. Hua, G. Ramakrishnan, G. Qi, L. Gao, and Y.-F. Li, "Difficulty-controllable multi-hop question generation from knowledge graphs," in *International Semantic Web Conference*. Springer, 2019, pp. 382–398.

[84] N. F. Liu, T. Lee, R. Jia, and P. Liang, "Can small and synthetic benchmarks drive modeling innovation? a retrospective study of question answering modeling approaches," *arXiv preprint arXiv:2102.01065*, 2021.

[85] M. Bartolo, T. Thrush, R. Jia, S. Riedel, P. Stenetorp, and D. Kiela, "Improving question answering model robustness with synthetic adversarial data generation," *arXiv preprint arXiv:2104.08678*, 2021.

[86] A. D. Lelkes, V. Q. Tran, and C. Yu, "Quiz-style question generation for news stories," in *Proceedings of the Web Conference 2021*, 2021, pp. 2501–2511.

[87] S. Gupta, A. Agarwal, M. Gaur, K. Roy, V. Narayanan, P. Kumaraguru, and A. Sheth, "Learning to automate follow-up question generation using process knowledge for depression triage on reddit posts," *arXiv preprint arXiv:2205.13884*, 2022.