mathematics

MDPI

Article

ReqGen: Keywords-Driven Software Requirements Generation

Ziyan Zhao [1], Li Zhang [1], Xiaoli Lian [1,*], Xiaoyun Gao [1], Heyang Lv [1] and Lin Shi [2]

[1] The State Key Laboratory of Software Development Environment (SKLSDE), Beihang University, Beijing 100191, China
[2] Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
* Correspondence: lianxiaoli@buaa.edu.cn

Abstract: Software requirements specification is undoubtedly critical for the whole software life-cycle. Currently, writing software requirements specifications primarily depends on human work. Although massive studies have been proposed to speed up the process via proposing advanced elicitation and analysis techniques, it is still a time-consuming and error-prone task, which needs to take domain knowledge and business information into consideration. In this paper, we propose an approach, named *ReqGen*, which can provide further assistance by automatically generating natural language requirements specifications based on certain given keywords. Specifically, *ReqGen* consists of three critical steps. First, keywords-oriented knowledge is selected from the domain ontology and is injected into the basic Unified pre-trained Language Model (UniLM) for domain fine-tuning. Second, a copy mechanism is integrated to ensure the occurrence of keywords in the generated statements. Finally, a requirements-syntax-constrained decoding is designed to close the semantic and syntax distance between the candidate and reference specifications. Experiments on two public datasets from different groups and domains show that *ReqGen* outperforms six popular natural language generation approaches with respect to the hard constraint of keywords' (phrases') inclusion, BLEU, ROUGE, and syntax compliance. We believe that *ReqGen* can promote the efficiency and intelligence of specifying software requirements.

Keywords: software requirements generation; knowledge injection; requirements syntax

MSC: 68N01; 46-04

Citation: Zhao, Z.; Zhang, L.; Lian, X.; Gao, X.; Lv, H.; Shi, L. ReqGen: Keywords-Driven Software Requirements Generation. *Mathematics* **2023**, *11*, 332. https://doi.org/10.3390/math11020332

Academic Editor: Ioannis G. Tsoulos

Received: 13 December 2022
Revised: 30 December 2022
Accepted: 3 January 2023
Published: 9 January 2023

1. Introduction

There is no doubt about the importance of software requirements to the whole software life-cycle [1–3]. As the vital product of requirements analysis, software requirements specifications act as the essential bridge between the requirements analysis stage and the following development and testing. Currently, writing software requirements specifications primarily relies on human work, and this work is complex and time-consuming due to the following factors:

- Enough domain knowledge is required to state the right content, as well as to select appropriate words and expressions. However, domain analysis typically requires non-trivial human effort [4].
- Writing the specifications word-by-word is time-consuming, let alone that many expressions are repeated, especially in similar or related requirements. Writing or locating, copying, and then pasting the repeated content is a waste of human effort.
- Generally accepted requirements syntax, such as EARS [5], is suggested for writing well-formed specifications. Learning and carefully applying non-business-related knowledge is also a burden on requirements analysts.

Intuitively, it would be quite helpful to automatically recommend requirements specifications as long as the analysts can provide some simple related information almost

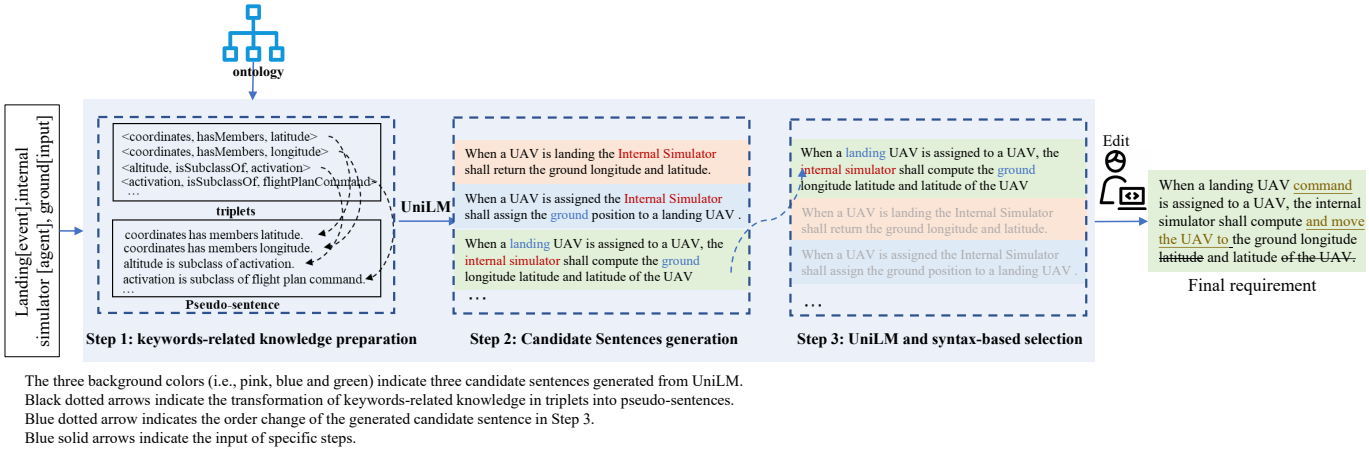
effortlessly. To the best of our knowledge, most of the automated requirements specifications generation work focus on transforming software engineering models (e.g., the i* framework [6,7], KAOS [8–10], UML models [11–13]) or other semi-structured inputs (e.g., security goals in specific syntactic patterns [14]) into natural language requirements specifications based on pre-defined rules, which are usually brittle and restrict the usability of these approaches. What is more, constructing expressive and precise models is another complex work as well.

Besides, much research has been conducted to speed up the requirements analysis process by identifying and analyzing requirements information from, for example, domain documents [4,15–17], developers' online chats [18], and product descriptions [19,20]. The primary results of these studies are features [19–25], requirements-related sentences [4,15,16,18,26–29], and requirements classifications [22,30,31]. Although these resources are relevant and helpful for requirements acquisition and generation, they are only *separate pieces of information*. Requirements analysts still need to spend significant efforts to understand them, to integrate them with the project background, and then, to specify the final requirements by following at least one requirements syntax [16].

In this work, we aim to automatically generate the requirements statement draft once the analyst has an intuitive idea and can provide two or more keywords (phrases) of the desired requirements. Optionally, the analyst can also suggest the syntax roles of part of or all keywords. We hope to recommend the requirements specifications, although requirements analysts probably need to revise our generation for the final acceptance.

In this paper, we propose an approach, named *ReqGen*, to generate requirements specifications from the keywords (phrases) provided by requirements analysts. In particular, three critical designs are proposed based on the basic Unified pre-trained Language Model (UniLM) [32], which was selected as the backbone of our framework because of its promising performance in natural language generation (NLG) tasks [32–34]. Like other pre-trained language models, UniLM can adapt to downstream tasks smoothly through light fine-tuning [35]. What is more, its parameter sharing design makes the learned text representations more general since the corpus context is utilized in different ways during the pre-training procedure, which helps mitigate overfitting to any single LM task [32]. First, we injected domain knowledge in the format of pseudo-sentences created from domain ontologies into several selected layers of UniLM. Second, we used a copy mechanism in the training phase and set the given keywords as a hard constraint to ensure that they occur in the final statements. Last but not least, we designed a requirements-syntax-constrained decoding approach to accommodate the requirements syntax constraints of the generated statements, given that well-formed requirements should follow a certain syntax [5,36,37].

To illustrate our approach, we contrived one example from a requirements instance of ISO/IEC/IEEE 29148:2018(E) [37], shown in Figure 1. Given that analysts can provide three keywords (i.e., *landing*, *internal simulator, and ground*) and their syntax roles (i.e., *event*, *agent, and input*, respectively), our *ReqGen* will firstly retrieve the keywords-related knowledge from the domain ontology and convert them into pseudo-sentences in the first stage. Then, these pseudo-sentences will be injected into UniLM, and the candidate requirements sentences will be produced in the second stage. In the third stage, *ReqGen* sorts the candidate sentences based on the default scoring in UniLM and our syntax-constrained measure in decoding and selects the final requirements draft. Finally, requirements analysts can edit the draft to obtain the final requirements statement.



The three background colors (i.e., pink, blue and green) indicate three candidate sentences generated from UniLM.
Black dotted arrows indicate the transformation of keywords-related knowledge in triplets into pseudo-sentences.
Blue dotted arrow indicates the order change of the generated candidate sentence in Step 3.
Blue solid arrows indicate the input of specific steps.

Figure 1. One example showing the target of this work.

To evaluate our approach, we conducted experiments on public datasets consisting of two domains, which were collected from previous work [38]. The results demonstrate the promising performance of *ReqGen*, especially on the dataset with a larger knowledge scale. The comparisons with six popular NLG approaches show that *ReqGen* obtains better Bilingual Evaluation Understudy (BLEU), Recall-Oriented Understudy for Gisting Evaluation (ROUGE), and requirements syntax compliance. We also show the effectiveness of the three proposed components in *ReqGen* using an ablation experiment. The major contributions of this work are as follows.

- An approach, *ReqGen*, for automatically generating software requirements statements based on two or more keywords and their syntax roles.
- The evaluation of *ReqGen* on two public datasets from different domains with promising results and suggestions for knowledge injection into pre-trained models.
- The source code and experimental data made publicly available on Github https://github.com/ZacharyZhao55/ReqGen, accessed on 1 September 2022.

***Significance*:** Requirements recommendation is a major concern in RE. Comparing with the traditional approaches, we took a step forward by recommending the requirements specification draft rather than only identifying the useful, but indirect information.

2. Background

This section describes three key techniques: UniLM, attention mechanisms, and bidirectional long short-term memory (Bi-LSTM).

2.1. UniLM

We used UniLM [32] as the backbone of our approach because of its good NLG performance [33,39]. UniLM is a multi-layer Transformer network with 786 hidden dimensions and 16 attention heads. Its parameter size is 340M, and the activation function is the Gaussian error linear unit (GeLU), the same as those of the Bidirectional Encoder Representation from Transformers (BERT) [40]. It is pre-trained using two unidirectional language models (LM), one bidirectional LM, and one seq2seq LM. For our requirements generation task, we configured it using seq2seq LM (also known as the encoder–decoder model). UniLM uses the 12-layer bidirectional Transformer encoder like BERT. Each encoder layer includes a multi-head attention and a feed-forward neural network. During decoding, UniLM uses beam search (the beam size is five in our implementation) to select the candidate tokens of top-k scores in each step.

2.2. Attention Mechanism

The model with conventional encoders cannot pay attention to information outside the input sequence. Our aim was to make the model able to pay more attention to the

injected knowledge, in other words to let the model change the weights of the original hidden state so that higher weights can be assigned to the injected knowledge.

Many attention methods exist, including the dot product model, scaled dot product model, additive model, and bi-linear model. Moreover, the steps of these methods are almost the same, including: (1) the calculation of the similarity between a *Query* and *Key* and obtaining the weights; (2) normalization of the weights; and (3) summation of the weights with a *Value*. The scaled dot product is the most-common, fastest, and most-space-efficient attention mechanism [41], and we used it. This model has an input consisting of a *Query* and *Key* with d_k dimensions and a *Value* with d_v dimensions. We first calculated the dot product of the *Query* and *Key* (MatMul), then divided every *Key* by $\sqrt{d_k}$ (Scale). Finally, we fed it to a *softmax* layer to obtain the weights corresponding to the *Value*s.

2.3. Bi-LSTM

To better understand the injected knowledge, contextual information should be considered in the model. Before injection, we need to encode the knowledge. In our case, we selected the ontology as the source of the supplemented knowledge for keywords. The domain ontology is a graph structure, and most entities have both parent and child nodes. In other words, the contextual information of the injected knowledge from the ontology is bidirectional, not sequential. Therefore, in the method proposed in this paper, to obtain a more comprehensive bidirectional representation of the injected knowledge, we used Bi-LSTM [42] to perform the encoding, where the knowledge is embedded by BERT.

Bi-LSTM is a variant of the recurrent neural network, which combines two standard LSTM [43] layers in opposite directions to learn the two-way representation. An LSTM cell has three gates: the input, forget, and output gates. These three gates have different functions, acting as filters that decide which information to keep and which to forget. This increases the accessible information of the Bi-LSTM network, and hence, the model can better understand the context of the knowledge.

3. Related Work

We first review the requirements generation research. Then, we describe the related work on common NLG.

3.1. Automated Requirements Generation

The existing studies on automated requirements generation mainly focus on transforming the (semi-)structured models (e.g., business process model in [44,45], i* framework in [6,7], KAOS and Ojectiver in [8–10], UML models in [11–13]) or other representations (e.g., security goals in [14]) into specific syntactic pattern-oriented natural language requirements specifications, based on a set of pre-defined rules. They usually require precise representation of the critical elements, such as the roles, inputs, outputs, and their relationships in the business process model [44] or the security goals expressed as clauses with the main verb + several security criteria + several target assets [14]. Both the (semi-) structured inputs and the pre-defined rules restrict the application scope of these approaches.

Mohamed et al. [46] proposed to generate *non-quality* requirements based on grammatical rules and a supplied dictionary, to resolve small-scale requirements sets for current requirements quality checking work.

There are very few studies on requirements specification generation from a few simple keywords. Most available research on software requirements capturing and generation concerns automatically collecting and/or analyzing requirements-related information with the purpose of *assisting* requirements analysts to *manually* acquire, interpret, and specify the final requirements statements. This information can be requirements-relevant sentences obtained from domain documents [4,15–17,26,30], features and their relationships mined from developer online chats [18], online reviews [23,24] and product descriptions [19,20], or the classifications of types of user statements [22,31] and of obligation or non-obligation statements in contracts [30]. Besides, better product descriptions can be automatically

generated with the techniques of summarization from user reviews [47,48] and website information [49]. Although these sources of information are definitely essential for requirements specification, analysts still need to invest significant time and effort in interpreting this information and specifying the requirements clearly under the premise of knowing the basic requirements syntax and the domain phrases involved in the requirements.

3.2. Automated NLG with Lexical Constraints

In an early work on lexical constraints generation, Mou et al. [50] proposed the backward and forward language model (B/F-LM) and used the recurrent neural network to generate previous and subsequent words conditioned on the given word. Liu et al. [51] extended the B/F-LM by introducing a discriminator, but these two methods can generate sentences with only one lexical constraint. To overcome this limitation, Hokamp et al. [52] incorporated constraints by performing a grid beam search in the sentence space. The CGMH framework [53] models local transitions (e.g., deletion and insertion) to achieve better fluency, but it is slow to converge. Sha et al. [54] proposed an unsupervised lexical constraint generation method, in which a series of differentiable loss functions was used to calculate the fluency of the generated sentences and to determine whether they satisfy the constraints. Ding et al. [55] proposed a framework to customize the message content for appealing to different individuals.

Recently, the fine-tuning of pre-trained language models has provided more research opportunities for small datasets in many domains. BERT [40] and RoBERTa [56] use masked language modeling pre-training objectives for deep bidirectional representations by jointly conditioning on both the left and right contexts in all layers. GPT-2/3 [57,58] and CTRL [59] are causal language models, which use the auto-regressive language to model the target. UniLM [32] and GLM [60] combine the advantages of the first two models and use a special mask mechanism so that the tokens in the input sequence can focus on each other, whereas the tokens in the output sequence can focus only on the tokens to the left. MASS [61], T5 [62], and BART [63] are encoder–decoder models and adopt the standard Transformer structure [41].

4. Our Approach: *ReqGen*

As shown in Figure 2, we selected UniLM as the backbone of *ReqGen*, because of its advantages in natural language understanding and NLG [32]. On the basis of UniLM, four components were designed and implemented to improve the compliance of the generated statements with domain knowledge and software requirements syntax (as indicated by the light blue rectangles in Figure 2):

- A knowledge preparation module that retrieves keywords-related information from the domain ontology (in terms of triples consisting of entity pairs and their relationships), transforms all related information into pseudo-sentences, and selects the knowledge to be injected into the different layers of UniLM.
- A knowledge-injection model to inject the knowledge produced by the knowledge preparation module into UniLM. The injected knowledge sequence is encoded by a Bi-LSTM structure using a BERT-based sentence embedding and is injected through the attention mechanism.
- A keywords copy mechanism was added in the original UniLM, which enables it to perform a copied-word classification task in the UniLM training. Moreover, a prediction method was added to the UniLM inference model to decide whether the next token is a copied word.
- A requirements-syntax-constrained decoding module that includes a semantics-related metric used in the inference model with the original beam search to optimize the final statements towards a specific syntax.

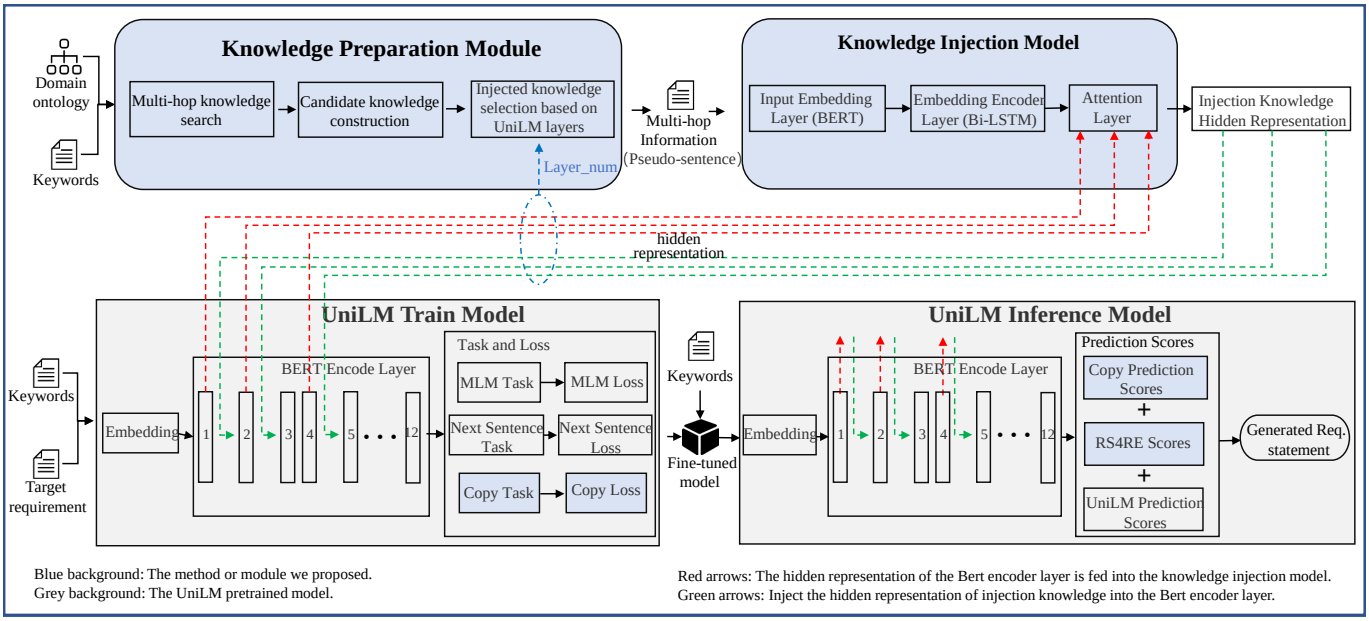


Figure 2. The procedure of our *ReqGen*.

4.1. Knowledge Preparation Module

The purpose of this module is to produce the knowledge to be injected into the backbone UniLM according to the input keywords from the domain ontology. It consists of three steps: multi-hop knowledge search, pseudo-sentence construction, and knowledge selection for the UniLM layers.

4.1.1. Multi-Hop Knowledge Search

This step aims to obtain the keywords-related concepts and their relationships from a domain ontology. We would like to increase the probability of the co-occurrence of these concepts and the input keywords by managing the attention in the knowledge injection model.

We obtained knowledge from the domain ontology, which is a collection of all of the relevant concepts and their relationships in a single domain. It is represented as a graph structure composed of triples $\langle Entity_1, Relation, Entity_2 \rangle$, built based on the OWL ontology language rules [64].

To obtain the context information of the keywords, we first performed a multi-hop graph search to acquire as much useful information as possible (five hops in this work), starting from the keywords, and collecting all types of entities in the retrieved paths.

Because all of the input keywords are regarded as start nodes, there must be repeated retrieved paths. We believe the concepts in the repeated paths are important; that is, the more often they are retrieved, the more important they are. Following this principle, we filtered some concepts according to the retrieval times to avoid noise.

4.1.2. Pseudo-Sentence Construction

We converted the extracted multi-hops $\langle Entity_1, Relation, Entity_2 \rangle$ into pseudo-sentences for the following injection task. In the OWL ontology language, entity types include *Classes*, *Object Properties*, and *Named Individuals*. There are two types of relations between these entities: the *subclass relation* and *constraint relation*. We set pseudo-sentence generation rules for these two relation types as follows:

(1) The *subclass* type includes *subClassOf* and *subPropertyOf* relations:

- *subClassOf* or *hasSuperClasses*: For the triple of $\langle a, subClassOf, b \rangle$, we created a sentence of "a is subclass of b". We converted the triple of $\langle \mathtt{a}, \mathtt{hasSuperClasses}, \mathtt{b} \rangle$ into "a has super class b."

- *subPropertyOf*: For the triple of ⟨a, subPropertyOf, b⟩, we created a sentence like "a is subproperty of b".

(2) The *Constraint* type includes *has domain* and *has range* relations:

- *has domain*: specifies the domain of a property *P*, indicating that any resource with a given property is an instance of the domain class (e.g., ⟨teaching, hasdomain, teacher⟩).
- *has range*: specifies the range of a property *P*, indicating that the value of a property is an instance of the range class (e.g., ⟨teaching, hasrange, lesson⟩).

Generally, the relationships of *has domain* and *has range* are paired. If the property has a *has domain* triple, there must be a corresponding *has range* triple simultaneously. Hence, we produced sentences for the relationship pairs. Taking the triples of ⟨teaching, hasdomain, teacher⟩ and ⟨teaching, hasrange, lesson⟩ as examples, we can create the pseudo-sentence "Teacher is teaching lesson".

Note that the grammar of the pseudo-sentences generated automatically may be wrong. However, it can provide an important context for the concepts in sentences that is required by the Bi-LSTM encoder.

4.1.3. Knowledge Selection towards the UniLM Layers

In the human learning process, when learning new things, we typically learn relatively broad knowledge first and, then, pay more attention to the essential parts, which is a repeated and gradual process. Inspired by this, we injected different keywords-related knowledge three times into the UniLM encoder, and the injected knowledge was more refined every time, rather than the common one-time knowledge injection into the pre-trained model [39].

Following the traditional one-time knowledge injection into the first layer of the pre-trained BERT model [39], we observed that the injected knowledge cannot be reflected in the decoder of BERT. We believe the primary reason is the strong fitting ability of BERT on training data, i.e., too small weights of the external knowledge injected once compared to those of the training data.

Therefore, we designed a knowledge injection mechanism by increasing the number of injections and selecting different knowledge for different layers of UniLM. In this work, we injected the knowledge into the 1st, 2nd, and 4th layer of the UniLM encoder, as shown in Figure 2. Moreover, we injected more valuable information into the higher layers. In particular, we selected all 5-hop keywords-related information for the first layer, 2-hop information for the second layer, and only 1-hop information for the fourth layer. This gradual refinement of the injected knowledge should enhance the ability of UniLM to absorb externally injected information. The evaluation and analysis of these three layers' selection (i.e., 1, 2, 4) are shown in Section 5.2.2.

4.2. Knowledge Injection Model

As Figure 2 shows, the knowledge injection model is composed of three layers. The multi-hop information in the format of pseudo-sentences is first embedded using BERT and, then, encoded using Bi-LSTM. Finally, an attention layer is used to increase the weights of the injected knowledge in the hidden representation layer of the UniLM encoder.

4.2.1. Pseudo-Sentence Embedding and Encoding

The basic unit of BERT embedding is a token. Thus, during the embedding phase, a pseudo-sentence set (knowledge) is first given to the BERT tokenizer. The [CLS] and [SEP] labels are added at the beginning and end of each sentence, respectively. Moreover, the tokenizer separates words into sub-words including the root of the words (e.g., "flying" is divided into ["fly","##ing"]).

There are three parts of the BERT embedding process: token embedding, position embedding, and segment embedding. Token embedding converts the tokens into vector representations in 768 dimensions ([1, *n*, 768], where *n* is the length of a sequence). Segment embedding is used to distinguish between the keywords and target requirements in each

input pair ([1, n, 768]). Position embedding converts the position information in the sequence into a vector representation of 768 dimensions ([1, n, 768]). Then, we summed these three embeddings and used this representation as the output of the BERT embedding.

We sent the embedding to the Bi-LSTM encoder [42] to obtain the pseudo-sentence encodings. The shape of the encoded hidden representation is [b, n, 768], where b is the batch size of UniLM and n is the sequence length.

4.2.2. Attention Mechanism

We used an attention mechanism to emphasize the importance of the injected knowledge. An attention mechanism is formally defined as follows:

$$\begin{aligned} A &= softmax(QK^T/\sqrt{d} + M) \\ H_{ctxt} &= A \cdot V \\ H_{knowledge} &= LayerNorm(W^T \cdot H_{ctxt} + H_{UniLM}) \end{aligned} \tag{1}$$

Here, Q is a Query, which represents the hidden representation of the UniLM encoder layer; K and V are the Key and Value, respectively representing the hidden representations of the pseudo-sentences (knowledge) encoding. To obtain the attention weights of Q and K, we used the scaled dot product method. M is the joint mask of the pseudo-sentence and the original input of UniLM (i.e., source keywords and target specifications), and d represents the length of the last dimension of K. According to experience, the use of $\sqrt{d}$ reduces the sensitivity of the method to the length of K and improves the stability of network training. Then, we calculated the dot product between A and V to update V and obtain the context representation of the attention. Finally, we summed the updated values H_{ctxt} and H_{UniLM} and fed them into *LayerNorm* (i.e., the normalization layer) to obtain the hidden representation of the knowledge.

4.3. UniLM Module

We modified UniLM in three ways, by adding multi-layer knowledge injection, a copy mechanism in the training model, and a requirements-syntax-constrained decoding in the inference model. The knowledge injection was discussed in Sections 4.1 and 4.2. Here, we focus on the other two modifications.

4.3.1. Copy Mechanism

The copy mechanism is very popular in NLG tasks such as translation and conversation generation [65]. Its aim is to resolve the "hard-constraint" problem, which requires the tokens or fragments of the source sequence to occur in the target sequence. However, most copy mechanisms can only guarantee the copy of a single word, not continuous fragments [66]. However, in our scenario, the input keywords (phrases), even the single-term words, can be multi-term words after tokenization, making it a challenge to ensure the integrity of words in the generated statements.

We adopted a simple, but novel copy mechanism [66]. In our implementation, this mechanism first marks the copied fragments in the target requirements according to the source keywords. Then, we added a new copied label prediction task in the training phase of our seq2seq model to predict whether a token has been copied from the source keywords or not. In the decoder prediction stage, the original next token prediction probability is changed to a mixed probability model of the next token prediction probability and copied token prediction probability.

4.3.2. Requirement-Syntax-Constrained Decoding

Well-formed requirements specifications should follow a certain syntax, and there are several related guidelines or models, such as EARS [5], IEEE 29148:2018 [37] and M-FRDL [36]. They all define the fine-grained elements of single requirements specifications, and each indicates one semantic role.

The requirements generated by *ReqGen* should attempt to follow at least one existing syntax so that they can accurately capture stakeholder needs [37]. Hence, if requirements analysts follow a specific requirements syntax when writing requirements specifications and they could give the semantic roles that the keywords belong to (the content in brackets of Figure 1), we would like to use this information to improve the generated sentence further. We designed an indicator called *RS4RE* to evaluate the overlap of the semantic constitution of the generated requirements with those set by engineers. The *RS4RE* value of each generated candidate statement is used to help select the final statement by adding it to the original probability score during beam search.

RS4RE is defined formally as follows. We measured the closeness of agreement of each semantic element in the generated statement and the input of an analyst.

$$RS4RE = \sum \alpha_i \frac{|E_i(R_{aut}) \cap E_{i\_ref}|}{|E_{i\_ref}|} \quad where\ 1 \leq i \leq N \tag{2}$$

Let N be the number of semantic elements in the syntax requirements analysts have selected and E_i be the i_{th} element. Here, E_{i_ref} is used to indicate the word set of the element E_i given by the requirements analysts, and $E_i(R_{aut})$ indicates the set of words of the element E_i in the automatically generated sentence. The agreement on element i is calculated as the ratio of the scale of overlapping words to the size of the manually given set. Moreover, α_i is a hyper-parameter that indicates the weight of element i, and the sum of all α_i is 1.

In this work, we used M-FRDL [36] as the syntax because they shared the source code with us. This enabled us to automatically identify the fine-grained elements in the natural language requirements, which is required by the automated calculation of *RS4RE*.

5. Experimental Evaluation

We evaluated *ReqGen* by addressing the following research questions:

- **RQ1:** How well does *ReqGen* perform in comparison with existing NLG approaches on requirements specification generation based on keywords?
- **RQ2**: To what extent does the multi-layer knowledge injection contribute to the requirements specification generation?
- **RQ3**: To what extent does the knowledge frequency filtering contribute to the requirements specification generation?
- **RQ4**: To what extent does each proposed design component contribute to the requirements specification generation?

5.1. Experimental Design

5.1.1. Data Preparation

To evaluate our *ReqGen*, we used two datasets of open-source requirements specifications and two domain ontologies, i.e., the unmanned aerial vehicle (UAV) and building automation system (BAS) domains, following previous work [38].

The UAV requirements are from the University of Notre Dame (https://dronology.info/, accessed on 23 May 2018) including 99 requirements [67]. The public ontology for the UAV domain (http://www.dronetology.net/, accessed on 2 February 2021) includes 400 entities. The BAS requirements are from the Standard BAS Specification (2015) [68] and consist of 456 requirements, involving functional, performance, and security requirements. The open-domain model of BAS (https://gitlab.fi.muni.cz/xkucer16/semanticBMS, accessed on 1 December 2017) includes 484 entities.

The requirements are represented using natural language sentences, and we required the related keywords before the model training and testing. Thus, we performed a reverse extraction process. To be specific, we automatically extracted the noun words or phrases based on the parts-of-speech tagged by Stanford CoreNLP [69]. We extracted the noun and noun phrases with a series of linguistic filters for the nested noun selection, such as

$Noun^{+}Noun$. For each requirements, we randomly selected n noun phrases ($n \in [2, N]$, where N is the total number of noun and noun phrases in one requirement) as the keywords.

To evaluate the effectiveness of the requirements-syntax-constrained decoding described in Section 4.3.2, we implemented the M-FRDL constraint [36] and manually assigned the semantic roles to the randomly selected keywords according to their context in the requirements statements. To ensure correctness, we invited one author of the work [36] to check and revise them.

Another problem with the data is that the domain ontology may be unable to cover all the content of the requirements because these datasets were generated by different groups who have different concerns. Only if the keywords are contained in the domain ontology will the ontology be helpful for the keywords-driven requirements generation. Thus, we performed automated domain ontology completion using an approach of [70]. This approach first aligns the requirements and domain ontology using TransE [71]. Then, it selects the requirements concepts that are related to the entities in the domain ontology and adds them, as well as their corresponding relationships to the ontology.

Given the limited data, we performed 10-fold and 5-fold cross-validation on the UAV and BAS requirements, respectively. In addition, we performed a batch knowledge injection and injected the knowledge for all of the random keywords of the testing requirements, considering the potential association of the requirements. In particular, 17,681 pseudo-sentences were created for the 10 test requirements of the UAV and 139,494 pseudo-sentences for the 93 test requirements of the BAS.

5.1.2. Baselines

We selected six popular constrained text generation approaches as the baseline methods, including four pre-trained models and two other NLG models (i.e., POINTER [72] and CGMH [53]). Similarly, we trained these models using the UAV and BAS datasets separately (10-fold and 5-fold cross-validation, respectively):

- BERT [40] jointly conditions on contextual information, which enables pre-training and deep bidirectional representation from unlabeled text. We fine-tuned the BERT base model on our task, and the number of parameters was 110 M.
- Generative Pre-trained Transformer 3 (GPT3) [58] uses the one-way language model training of GPT2. The model size was increased to 175 billion, and 45 terabytes of data were used for training. GPT3 can perform downstream tasks without fine-tuning in a zero-shot setting. We also did not perform fine-tuning.
- Bidirectional and Auto-Regressive Transformers (BART) [63] is a denoising autoencoder built with a sequence-to-sequence model suitable for various tasks. It uses a standard Transformer-based neural machine translation architecture. It is trained using text corrupted with an arbitrary noising function and by learning to reconstruct the original text. We used the BART-large model on our task, and the number of parameters was 400 M.
- UniLM [32] uses three types of language modeling tasks for pre-training, which is achieved by employing a shared Transformer network and using specific self-attention masks to control the context of the prediction conditions. We used the UniLM base model, whose parameter amount is 340 M.
- Zhang et al. [72] proposed POINTER, which is based on the inserted non-autoregressive pre-training method. A beam search method was proposed to achieve log-level non-autoregressive generation.
- The constrained generation by Metropolis–Hastings sampling (CGMH) method [53] can cope with both hard and soft constraints during sentence generation. Different from the traditional latent space usage, it directly samples from the sentence space using the Metropolis–Hastings sampling.

5.1.3. Metrics

We selected *BLEU* [73] as the first metric, which is commonly used in machine translation, NLG, and source code generation. It measures the degree of overlap between the generated and reference sentences using n-grams. The higher the degree of overlap, the higher the quality of the generated text. Here, *BLEU1* measures word-level accuracy and *BLEU2* measures sentence fluency to a specific degree. In addition, we employed the *ROUGE* metric [74]. *ROUGE-N* calculates the total sum of the number of n-grams occurring in both the generated and target sentences, and *ROUGE-L* calculates the longest common subsequence. We calculated the recall, precision, and F-measure for each kind of ROUGE rather than the simple n-gram recall [74].

Besides, we used the *RS4RE* metric to evaluate the agreement of syntax compliance between the generated and target sentences, as described in Section 4.3.2. Here, E_{i_ref} refers to the set of words in the i_{th} semantic element of the target requirement. We also recorded the time used by each model (except GPT-3) for training and testing, considering its practical value.

5.2. Experimental Results and Analysis

5.2.1. RQ1 Effectiveness of Our Approach

We illustrate the experimental results for the UAV and BAS cases in Table 1. The average and standard deviation are given for each metric on the 5 and 10 runs on the UAV and BAS, respectively. For better illustration, we highlight the cells with the best results for each metric with grey background, and the second best results with yellow background. Table 2 presents an example of the requirements generated by each baseline and *ReqGen*, from which we can draw three conclusions:

(1) **Our method achieves the best or second-best results for all metrics.** In the UAV case, our method yields the best results for five metrics and the second-best results for the remaining eight metrics. BART achieves good performance with six best and five second-best results, which demonstrates the benefits of its hard constraint design. However, it can only embed a single input word and loses the complete semantic meaning of phrases (i.e., it obtains better BLEU1 and ROUGE-1 results, but weaker BLEU2 and ROUGE-2 results), which can also be seen in the example in Table 2. Moreover, *ReqGen* obtains a better RS4RE result than BART, indicating that it has a better semantic-oriented text generation ability.

On the BAS domain, our *ReqGen* achieves seven best and four second-best results, outperforming all baselines. The basic UniLM performs moderately well because it considers both the input and its context during the next token prediction. This is the reason that we select it as the backbone of *ReqGen*.

(2) **The four pre-trained models perform better than the other two baselines.** Among the four pre-trained models in the UAV domain, BART obtains the best ROUGE-1, ROUGE-2, and ROUGE-L results. It even outperforms *ReqGen* in ROUGE-1, which is possible because BART forces all keywords to be included in the output during decoding. However, because it only sees the input, it performs worse in ROUGE-2. UniLM obtains good BLEU1 and BLEU2 results and the best ROUGE-2 results in the BAS case, indicating its better ability to generate relatively fluent statements.

CGMH performs worst according to the two BLEU metrics and the F-measure of the three ROUGE metrics, as shown in Tables 1 and 2. POINTER also has a weak performance, and a possible reason for this performance could be that it places the entire generation burden on the decoder, which means that the sentences it generates are very long, but the correlation with the target is weak (see the example in Table 2).

From the standard derivation values, we can observe that the stability of the four pre-trained models is weaker than the other two baselines on the UAV. However, for the bigger BAS case, all of the pre-trained models achieve more stable results. This indicates that more domain knowledge is helpful for the stable results' generation for the pre-trained models.

(3) **Our method is slightly slower than UniLM and BART.** As shown in the last column of Table 1, we recorded the average time consumed by each method (in hours) to perform the 10-fold cross-validation on the UAV and the 5-fold cross-validation on the BAS data. We did not record the time used by GPT3 because it does not need extra fine-tuning [58]. We observed that *ReqGen* needs slightly more time than UniLM and BART, but less time than the other approaches, even though it uses extra knowledge injection, a copy mechanism, and requirements-syntax-constrained decoding.

Table 1. Results of the baselines and our *ReqGen* (%).

	Method	BLEU1	BLEU2	ROUGE-1			ROUGE-2			ROUGE-L			RS4RE	Time (HRS)
				R.	P.	F.	R.	P.	F.	R.	P.	F.		
UAV	Bert_base	37.92 ±3.3	23.53 ±2.54	54.33 ±2.29	42.04 ±3.61	47.40 ±2.33	31.32 ±2.37	24.88 ±3.25	27.73 ±2.57	51.29 ±2.77	39.90 ±3.38	44.88 ±2.68	8.00 [2] ±1.27	1.5
	GPT3	34.20 ±6.64	18.28 ±5.41	34.51 ±8.66	59.42 [1] ±11.39	43.66 ±9.39	16.49 ±6.77	14.82 ±8.05	15.61 ±7.12	31.02 ±8.39	56.22 ±10.64	39.98 ±8.89	1.31 ±0.26	-
	UniLM	34.27 ±8.95	17.83 ±5.69	55.15 ±10.99	41.53 ±13.26	47.38 ±11.68	18.21 ±6.71	14.44 ±8.42	16.11 ±7.19	49.73 ±10.27	37.71 ±12.42	42.89 ±10.92	2.54 ±0.14	1
	BART	42.66 ±7.90	23.99 ±9.17	71.75 ±6.23	52.12 ±6.16	60.38 ±4.85	37.86 ±8.95	26.90 ±7.70	31.45 ±8.16	66.07 ±7.20	47.79 ±6.56	55.47 ±5.89	5.76 ±1.45	1
	CGMH	12.07 ±2.63	1.84 ±0.56	35.06 ±4.45	17.86 ±9.99	23.66 ±6.06	5.48 ±0.92	2.47 ±2.95	3.41 ±1.39	32.42 ±4.24	16.52 ±9.34	21.89 ±5.75	0.00 ±0.00	22
	POINTER	17.26 ±3.63	2.46 ±0.98	24.15 ±3.86	38.66 ±3.58	29.73 ±3.72	2.83 ±1.61	5.20 ±1.09	3.67 ±1.26	19.92 ±2.60	32.07 ±2.76	24.58 ±2.80	4.60 ±1.87	1
	ReqGen	42.15 ±6.55	25.04 ±6.19	69.93 ±5.44	49.91 ±6.13	58.25 ±5.66	39.03 ±6.04	28.21 ±8.27	32.75 ±6.87	65.12 ±5.96	46.47 ±6.92	54.23 ±6.27	8.89 ±1.18	1.2
BAS	Bert_base	29.08 ±2.23	9.28 ±1.63	46.69 ±2.58	35.90 ±3.31	40.59 ±2.54	12.84 ±1.73	10.57 ±1.86	11.60 ±1.66	40.79 ±2.51	31.64 ±2.98	35.64 ±2.45	12.48 ±2.05	1
	GPT3	28.34 ±2.47	7.87 ±1.21	37.17 ±2.56	41.07 ±4.25	33.56 ±3.18	10.02 ±1.15	9.64 ±1.56	9.30 ±1.28	31.98 ±1.85	35.74 ±3.12	30.53 ±2.33	7.08 ±1.33	-
	UniLM	30.08 ±2.15	13.31 ±1.06	42.44 ±1.36	33.58 ±1.98	37.49 ±1.29	24.42 ±1.14	18.86 ±0.99	21.28 ±1.01	39.58 ±1.41	31.47 ±1.22	35.06 ±1.16	13.56 ±2.43	0.8
	BART	27.05 ±1.63	10.60 ±0.55	63.67 ±0.36	37.37 ±3.93	47.10 ±1.38	20.59 ±0.56	12.43 ±1.64	15.50 ±0.77	58.78 ±0.34	34.14 ±3.73	43.19 ±1.39	14.58 ±1.79	0.7
	CGMH	10.58 ±2.25	1.85 ±0.32	57.69 ±4.33	20.94 ±11.21	30.73 ±5.99	4.23 ±0.55	1.64 ±1.41	2.36 ±0.77	44.70 ±3.06	16.41 ±8.05	24.01 ±4.26	3.26 ±0.84	72
	POINTER	21.34 ±3.74	2.34 ±0.75	25.34 ±5.05	43.82 ±3.36	32.11 ±4.02	1.93 ±1.43	3.69 ±0.74	2.53 ±0.96	21.01 ±3.74	36.53 ±2.59	26.68 ±3.07	5.72 ±0.57	1
	ReqGen	38.07 ±1.62	15.62 ±0.87	58.96 ±1.33	44.63 ±3.21	50.80 ±1.95	20.35 ±1.35	16.45 ±2.11	18.19 ±1.62	52.82 ±1.16	40.17 ±2.76	45.63 ±1.65	17.42 ±2.83	0.9

[1] Grey background: Best results for each metric. [2] Yellow background: Second-best results for each metric.

Table 2. Examples showing the requirements generated by the baselines and our method from three keywords.

Keywords: landing, internal simulator, ground **Target:** When given a **landing** [1] command the **internal simulator** shall move the UAV from to the **ground** altitude corresponding to its current longitude and latitude.		
Alg.	**Generated Sentence**	**Required Modifications**
Bert_base	when a flight **simulator** is activated the flight **simulator** shall compute the location of the UAV.	Useless: no semantic overlap with the target.
GPT3	When a UAV lands the **Internal Simulator** shall record the time of **landing**.	Useless: no semantic overlap with the target.
UniLM	When a UAV is loaded the **internal simulator** shall display the location of the UAV.	Useless:no semantic overlap with the target.
BART	**landing internal** flight **simulator** shall compute the **ground** position of a UAV.	The trigger is missing, and the main clause is partially right: when given a landing command [2], the internal flight simulator shall compute the ground position of a UAV and move the UAV to ground latitude.
CGMH	**simulator** owners for takeoff at a corresponding latitude.	Useless: syntax errors and no semantic overlap with the target.
POINTER	in which a single point **landing** system, when a single point that is assigned a separate GPS system or for an active navigation system has to orient at its current position and to display the current **ground** coordinates.	Useless: syntax errors and no semantic overlap with the target.
ReqGen	When a **landing** UAV is assigned to a UAV, the **internal simulator** shall compute the **ground** longitude latitude and latitude of the UAV.	The trigger is right, and the main clause is partially right: when a **landing** UAV command is assigned to a UAV, the **internal simulator** shall compute the **ground** longitude latitude and latitude of the UAV and move the UAV to ground latitude.

[1] Bold font: Keywords in the sentences generated by the all automated approaches. [2] Blue font: The parts added by analysts manually.

5.2.2. RQ2 Effectiveness of Multi-Layer Knowledge Injection

We also evaluated the effectiveness of the multi-layer knowledge injection in the two domains. To implement and evaluate *ReqGen*, given that we collected five-hop keywords-related information and there are 12 layers in UniLM, we had to determine (1) *which levels should have knowledge injected* and (2) *what kind of knowledge should be injected into the different layers*. Because of the uncertainty in the injected layers and the injected knowledge for one specific layer (i.e., which hop in the five hops), there are many combinations that could have been used in our experiments. Thus, we pruned the candidate combinations for the experiments.

As for the injected layers, Li et al. [39] injected knowledge into the first layer. Jawahar et al. [75] experimented with different layers of BERT on 10 sentence-level detection tasks. They observed that the 1st and 2nd layers learn surface information such as word detection in sentences. Layers 4 to 7 learn syntactic information such as word order sensitivity. Layers 8 to 12 learn semantic-level information such as subject–verb agreement. Inspired by these two studies, we focused on three combinations of shallow layers, shallow + syntactic layers, and shallow + syntactic + semantic layers, with layer combinations of (1,2), (1, 2, 4), and (1, 2, 4,8), respectively. Moreover, considering the five hops in our collected information, we made an extra evaluation on the combination (1, 2, 4,8,11), and each layer was assigned one-hop information.

For the injected knowledge, we followed the cognitive process of human learning, in which humans usually start from general and overview knowledge and, then, pay close attention to the critical parts. Similarly, we assigned more keyword knowledge to the lower layers. For example, for the layer combination (1,2), we assigned all 5-hop knowledge to the first layer and the most-related 1-hop knowledge to the second layer. The injected knowledge details of different layer combinations can be found in the *layer (hop)* of Table 3, which presents the results of this experiment.

For the UAV case, we observed that the best BLEU1 and BLEU2 scores are achieved by the (1, 2, 4) combination. For ROUGE-1, the best recall is achieved by injecting knowledge

into Layers 1, 2, 4 and 8, the best precision is achieved by the (1,2) combination, and the best of F-measure is achieved by the (1, 2, 4) combination. Similar phenomena were observed for the ROUGE-2 and ROUGE-L metrics. In the BAS case, the best BLEU1 and BLEU2 results are also achieved by the (1, 2, 4) combination. For ROUGE-1, the best F-value is achieved by the (1,2) combination; however, for ROUGE-2 and ROUGE-L, the best performance is achieved by the (1, 2, 4) combination. Besides, the stability of this configuration has no obvious change.

In summary, the (1, 2, 4) combination with the 5-, 2-, and 1-hop knowledge injection achieves seven best results in the UAV case and nine best results in the BAS case, out of all 11 metrics. Hence, we suggest three knowledge injections by injecting all 5-hop knowledge into the first layer, 2-hop knowledge into the second layer, and the most-critical 1-hop knowledge into the fourth layer.

Table 3. The effectiveness of different injection times (%).

	Layer (Hop)	BLEU1	BLEU2	ROUGE-1			ROUGE-2			ROUGE-L		
				R.	P.	F.	R.	P.	F.	R.	P.	F.
UAV	1(5), 2(1)	43.23 ±5.76	24.63 ±6.18	63.21 ±5.44	50.51[1] ±4.69	56.15 ±4.18	34.90 ±7.02	27.61 ±6.00	29.86 ±6.15	58.83 ±5.25	47.04 ±5.80	51.12 ±4.96
	1(5), 2(2), 4(1)	43.87 ±6.07	25.05 ±6.89	65.69 ±6.82	50.34 ±6.54	57.00 ±6.15	35.59 ±9.57	27.18 ±7.04	30.82 ±7.98	61.13 ±7.60	46.86 ±6.95	53.05 ±6.91
	1(5), 2(3), 4(2), 8(1)	42.57 ±3.29	23.63 ±4.65	66.22 ±7.59	49.96 ±3.98	56.95 ±4.35	35.14 ±9.76	26.10 ±5.86	29.95 ±6.52	61.39 ±7.77	46.57 ±4.72	52.95 ±5.02
	1(5), 2(4), 4(3), 8(2), 11(1)	40.2 ±5.82	21.64 ±5.97	63.04 ±6.08	47.37 ±5.91	54.09 ±5.05	32.40 ±8.58	24.17 ±6.44	26.89 ±7.04	58.65 ±6.15	44.17 ±7.07	49.42 ±6.17
BAS	1(5), 2(1)	34.51 ±2.69	13.25 ±1.77	55.54 ±3.00	41.74 ±2.20	47.66 ±2.36	17.86 ±1.94	14.13 ±1.82	15.78 ±1.84	49.87 ±2.38	37.66 ±1.92	42.91 ±1.96
	1(5), 2(2), 4(1)	34.55 ±2.28	13.73 ±0.69	55.02 ±2.97	41.76 ±2.07	47.48 ±2.38	18.30 ±1.22	14.68 ±0.99	16.29 ±1.01	49.83 ±2.54	38.03 ±1.66	43.14 ±1.95
	1(5), 2(3), 4(2), 8(1)	33.26 ±2.37	12.92 ±1.20	54.51 ±1.72	40.64 ±2.53	46.56 ±2.14	18.13 ±0.99	14.19 ±1.24	15.92 ±1.14	49.21 ±1.31	36.92 ±2.08	42.19 ±1.80
	1(5), 2(4), 4(3), 8(2), 11(1)	32.33 ±2.05	12.31 ±1.12	54.27 ±2.99	40.16 ±2.43	46.16 ±2.55	17.38 ±1.49	13.77 ±1.23	15.37 ±1.30	49.02 ±2.57	36.55 ±2.14	41.88 ±2.23

[1] Grey background: Best results for each metric for each case.

5.2.3. RQ3 Effectiveness of Frequency Filtering in the Knowledge Search

We further evaluated the impact of different frequency sets in the knowledge search on the final requirements generation (as described in Section 4.1.1). We regarded more-frequent knowledge as more important, and our aim was to inject only important information into *ReqGen* to reduce noise pollution in the data.

To evaluate the impact of a frequency filter (a single dependent variable), we set the injected layer as a constant. We experimented with frequency thresholds of 0, 10, and 50 on all five-hop knowledge injected into the first layer. In other words, we injected all traversed entities, entities occurring more than 10 times, or those occurring more than 50 times in the ontology graph traversal. The results are shown in Table 4, and the best result of each metric is highlighted with bold font and grey background.

We observed different results for the two cases. For the UAV, no frequency filtering is best, followed by 10 and 50 frequency filters, whereas, for the BAS, the best performance is achieved with 10 frequency filters and with 50 frequency filters is the second best. There is no obvious difference between the stability of their results. This result led us to the following two observations. (1) Intuitively, the frequency set is strongly correlated with the scale of injected knowledge. In our case, the five-hop knowledge of the UAV domain includes 17,681 pseudo-sentences, and the BAS includes 139,494 sentences (approximately 7.9 times). (2) When there is a massive amount of information, less- and more-refined knowledge is more valuable. For example, 50 frequency filters are better than no filter in the BAS domain. However, when the information is small, all related knowledge is valuable

(e.g., in the UAV case, no filter is better than the 10 frequency filters, which is better than the 50 frequency filters). However, because of the limited available cases, we cannot give a criterion for frequency filter selection temporarily in this initial study.

Table 4. The effectiveness of knowledge frequency filtering (%).

	Frequency Filtering	BLEU1	BLEU2	ROUGE-1			ROUGE-2			ROUGE-L		
				R.	P.	F.	R.	P.	F.	R.	P.	F.
UAV	no	42.78 ±3.84	24.14 [1] ±5.23	66.54 ±6.54	49.85 ±4.57	57.00 ±4.25	35.51 ±7.55	26.59 ±5.37	30.41 ±5.87	61.76 ±6.55	46.45 ±5.08	53.02 ±4.79
	10	43.50 ±5.07	23.97 ±5.04	65.80 ±7.59	49.92 ±5.27	56.77 ±5.20	34.99 ±8.68	26.22 ±5.58	29.24 ±6.14	61.75 ±7.30	47.04 ±6.11	52.46 ±5.79
	50	40.47 ±5.80	21.39 ±6.09	61.87 ±4.00	46.58 ±4.71	53.15 ±4.27	32.05 ±7.58	23.63 ±6.10	27.21 ±6.60	57.90 ±4.34	43.70 ±4.94	49.80 ±4.58
BAS	no	34.15 ±1.90	13.37 ±1.03	54.75 ±1.99	41.30 ±1.10	47.08 ±1.28	18.01 ±1.22	14.49 ±1.05	16.06 ±1.03	49.41 ±2.02	37.52 ±1.34	42.65 ±1.48
	10	34.43 ±1.80	13.65 ±1.38	55.93 ±2.41	41.76 ±0.98	47.82 ±1.10	18.84 ±1.29	14.80 ±0.92	16.58 ±0.97	50.29 ±2.22	37.76 ±0.77	43.13 ±0.96
	50	34.33 ±1.83	13.45 ±1.31	55.68 ±1.64	41.69 ±0.96	47.68 ±1.22	18.65 ±1.59	14.66 ±1.07	16.41 ±1.28	50.28 ±1.87	37.83 ±1.09	43.18 ±1.39

[1] Grey background: Best results for each metric for each case.

5.2.4. RQ4 Ablation Experiment

The ablation experiment was designed to verify the effectiveness of each critical proposed component in our *ReqGen*: multi-layer injection (including frequency filtering), copy mechanism, and syntax-constrained decoding. Table 5 shows the results of *ReqGen* and its four variants. For the convenience of comparison, we indicate the values that are better than those in the previous row with an up-arrow.

Table 5. The ablation experiment (%).

	Settings	BLEU1	BLEU2	ROUGE-1			ROUGE-2			ROUGE-L		
				R.	P.	F.	R.	P.	F.	R.	P.	F.
UAV	Layer 1	42.78 ±3.84	24.14 ±5.23	66.54 ±6.54	49.85 ±4.57	57.00 ±4.25	35.51 ±7.55	26.59 ±5.37	30.41 ±5.87	61.76 ±6.55	46.45 ±5.08	53.02 ±4.79
	Layer 1, 2, 4	43.87 ±6.07 ↑ [1]	25.05 ±6.89 ↑	65.69 ±6.82	50.34 ±6.54 ↑	57.00 ±6.15	35.59 ±9.57 ↑	27.18 ±7.04 ↑	30.82 ±7.98 ↑	61.13 ±7.60	46.86 ±6.95 ↑	53.05 ±6.91 ↑
	Layer 1, 2, 4 + 10 Fre.	44.80 [2] ±4.00 ↑	25.12 ±5.23 ↑	65.35 ±6.64	52.18 ±4.26 ↑	58.03 ±4.22 ↑	35.48 ±8.98	28.11 ±5.62 ↑	31.37 ±6.66 ↑	60.95 ±7.35	48.93 ±5.94 ↑	54.28 ±5.56 ↑
	Layer 1, 2, 4 + 10 Fre. + Copy	42.15 ±6.55	25.04 ±6.19	69.93 ±5.44 ↑	49.91 ±6.13	58.25 ±5.66 ↑	39.03 ±6.04 ↑	28.21 ±8.27 ↑	32.75 ±6.87 ↑	65.12 ±5.96 ↑	46.47 ±6.92	54.23 ±6.27
	Layer 1, 2, 4 + 10 Fre. + Copy + Syntax cons. (*ReqGen*)	42.15 ±6.55	25.04 ±6.19	69.93 ±5.44	49.91 ±6.13	58.25 ±5.66	39.03 ±6.04	28.21 ±8.27	32.75 ±6.87	65.12 ±5.96	46.47 ±6.92	54.23 ±6.27
BAS	Layer 1	34.15 ±1.90	13.37 ±1.03	54.75 ±1.99	41.30 ±1.10	47.08 ±1.28	18.01 ±1.22	14.49 ±1.05	16.06 ±1.03	49.41 ±2.02	37.52 ±1.34	42.65 ±1.48
	Layer 1, 2, 4	34.55 ±2.28 ↑	13.73 ±0.69 ↑	55.02 ±2.97 ↑	41.76 ±2.07 ↑	47.48 ±2.38 ↑	18.30 ±1.22 ↑	14.68 ±0.99 ↑	16.29 ±1.01 ↑	49.83 ±2.54 ↑	38.03 ±1.66 ↑	43.14 ±1.95 ↑
	Layer 1, 2, 4 + 10 Fre.	37.99 ±1.61 ↑	15.57 ±0.07 ↑	56.30 ±2.27 ↑	45.05 ±1.60 ↑	50.05 ±1.67 ↑	19.58 ±1.29 ↑	16.57 ±0.85 ↑	17.95 ±0.99 ↑	50.60 ±2.14 ↑	40.68 ±1.27 ↑	45.10 ±1.38 ↑
	Layer 1, 2, 4 + 10 Fre. + Copy	38.41 ±2.51 ↑	15.06 ±1.13	59.47 ±3.23 ↑	44.93 ±2.34	51.18 ±2.82 ↑	19.46 ±1.83	15.75 ±1.00	17.41 ±1.38	53.30 ±2.70 ↑	40.48 ±2.00	46.01 ±2.40 ↑
	Layer 1, 2, 4 + 10 Fre. + Copy + Syntax cons. (*ReqGen*)	38.07 ±1.62	15.62 ±0.87 ↑	58.96 ±1.33	44.63 ±3.21	50.80 ±1.95	20.35 ±1.35 ↑	16.45 ±2.11 ↑	18.19 ±1.62 ↑	52.82 ±1.16	40.17 ±2.76	45.63 ±1.65

[1] Up-arrow: The value in one cell is better than that of the previous row for the same metric. [2] Grey background: Best results for each metric for each case.

In the UAV case, compared with the one-layer knowledge injection, the three injections into Layers 1, 2, and 4 yield an approximately 1% increase in BLEU1 and BLEU2. For the

ROUGE results, the improvement is primarily in the three precision scores, which are increased by 0.5%. However, the recall in ROUGE is slightly decreased, possibly due to some related knowledge loss in Layers 2 and 4. However, the F-measure values remain unchanged or become better with respect to the values obtained for the single-layer injection. In the BAS case, all metrics, including the BLEU and three ROUGE metrics, are improved by three-layer knowledge injection. This observation shows that the multi-layer injection design mainly improves the BLEU metric and the precision of ROUGE, indicating that this mechanism enhances the ratio of valid words in the final statements. In other words, although the single-layer injection can help the model obtain the knowledge, *the multi-layer injection is further helpful for knowledge absorption*.

The comparisons between the second and third rows in both the UAV and BAS domains show the impact of 10 frequency filters on Layers 1, 2, and 4. We can see that the two BLEU values in both cases are better with this frequency setting. For the UAV, the precision and F-measure of the three ROUGE metrics increase further. However, the recall values are weaker. This is expected because the filtering operation on a relatively small set of knowledge would probably lessen the number of useful clues, whereas, for the BAS, due to its larger scale, all metrics improve even after filtering. We note that *the 10 frequency filters indeed effectively reduce the noise*.

Figure 3 shows one example of the effects of frequency filtering. In this example, when no frequency filtering is used, the terms "loses" and "routing" in the domain ontology are injected into *ReqGen* and reflected in the generated requirement, showing that noise words are reflected in the lexical constitutions of the generated requirements, which then determine the semantic meaning. By contrast, the example with 10 frequency filters shows that our frequency filtering method helps reduce the noise in the injected knowledge.

Source: GCS, connected UAV, middleware
Target: The **GCS** shall report newly **connected UAVs** to the **GCS middleware.**
No frequency filtering: When the **GCS** loses the **connection** to a **UAV** the **UAV Middleware** shall transmit the routing to an **UAV**.
10 frequency filtering: When a **UAV** is **connected** via the **GCS middleware** shall transmit the **connection** to **UAV**.

Blue background: Keywords of the original input.
Pink background: The target requirement statement.
Yellow background: The sentence generated by our approach with no frequency filtering.
Green background: The sentence generated by our approach with 10 frequency filtering.
Bold font: Keywords in the target and generated sentences.

Figure 3. An example with or without frequency filtering.

The comparisons between the fourth and fifth rows of Table 5 for both domains show the influence of the copy mechanism. From the results for the UAV domain, we observed that six results, mainly ROUGE scores, improve and achieve the best values with this design, whereas the two BLEU scores fall approximately 2%. Similarly, the results for five metrics increase, and the remaining six decrease for the BAS domain. This shows that the copy mechanism is helpful for the *recall of the overlapping n-grams* in the generated sentences. However, because it was designed to guarantee that the given keywords must appear in the generated sentence, this actually increases the length of the generated sentences. In other words, it is highly likely to dilute the ratio of the overlapping n-grams in the whole statements, which decreases the BLEU scores.

We also show the influence of the copy mechanism with an example in Figure 4, and we can see that all keywords, including the multi-term phrases, can occur in the final sentence. Without this mechanism, some keywords are abandoned, in favor of the *more important injected knowledge* and the syntax learned during the pre-training process of UniLM.

Source: operator, trend log, setup information
Target: **Operator** shall be able to change **trend log setup information.**
No copy mechanism: **Operator** shall be able to use a **trend log** to share **trend logs** with other **operators**.
Copy mechanism: **Operator** shall be able to use a **trend log** for quick **trend log setup information.**

Blue background: Keywords of the original input.
Pink background: The target requirement statement.
Yellow background: The sentence generated by our approach with no copy mechanism.
Green background: The sentence generated by our approach with copy mechanism.
Bold font: Keywords in the target and generated sentences.

Figure 4. An example with or without the copy mechanism.

Finally, we added syntax-constrained decoding into the settings, and the results are listed in the fifth row of each case in Table 5. We can see consistent enhancements of the ROUGE-2 results for both cases and a positive effect on the BLEU2 in the BAS domain and no change to the BLEU2 result of the UAV domain. This indicates that *our syntax decoding is helpful for valid two-gram generation*, the semantic units with a larger granularity than single-term words.

6. Discussion

6.1. Threats to Validity

Construct validity mainly concerns the evaluation metrics. We used BLEU1, BLEU2, and the precision, recall, and F-measure of ROUGE-1, ROUGE-2, and ROUGE-L, which are the most-common metrics of NLG [33,39,54]. Moreover, to consider the special syntax constraints of well-formed software requirements [5,36,37], we designed the RS4RE metric, which measures the syntax distance between the automated generated and target requirements. We also recorded the training and testing time of the automated approaches for practical consideration.

Besides, the UAV and BAS ontologies were collected online by [38]. Their quality will impact the construct validity. However, both ontologies have been peer-reviewed, and there are series of follow-up peer-reviewed works (i.e., 3 for UAV [76–78] and 40 for BAS [79]). We believe the quality of the two ontologies can be guaranteed.

Internal validity concerns the validity of the causal relations between the results and our approaches. We reused the open-source code of UniLM, the backbone of our *ReqGen* and the six baselines, to ensure the accuracy of experiments. Another threat comes from the measure of RS4RE, which involves the manual annotation of the semantic elements of M-FRDL [36] for the case requirements. In the two sets of requirements, the UAV requirements were annotated by the work of [36], and we used their original annotations directly. For the BAS domain, our authors performed pair annotations, which were then carefully checked individually by the first author of [36]. We believe the annotations are trustworthy.

External validity concerns the generalization of the experimental results to other cases. In this initial study, we experimented with two public datasets from different domains with different scales. All of the requirements and ontologies were from different groups, reflecting different writing styles. However, because of the data limitations, we only performed 10- and 5-fold cross-validation on the UAV and BAS cases. Although the results are promising, more diverse data for further experiments will be needed in the future.

6.2. Implications

Based on our results, the primary implication is that, like traditional large-scale text-based fine-tuning, concept-net knowledge injection is beneficial for pre-trained models performing downstream tasks. This means that our approach is potentially helpful for other document generation, such as the description of software design. In addition, multiple knowledge injections into the pre-trained model are better than once, and the knowledge injected into higher layers should be more refined than that injected into the lower layers, just as in the human learning process. Our experiments show that injection of all 5-hop

knowledge into the first layer, 2-hop knowledge into the second layer, and only the most-relevant 1-hop knowledge into the fourth layer achieves the best results on both domains in our task (as shown in Table 3).

The second implication suggested by our study is that automated requirements generation requires related domain knowledge, but "more" does not always mean "better". In the BAS case, 10 frequency filters were better than no filtering. However, the injected knowledge is also critical because the general UniLM has to learn the domain knowledge for the valid requirements generation. Thus, for the small UAV case, no filtering is the best (as shown in Table 4).

Besides, our study has two practical impacts. First, *ReqGen* can assist analysts in generating the final correct requirements quickly since it can automatically generate the keywords-driven, domain-knowledge-restricted, and syntax-compliant requirements draft. As show in Figure 5, we used the example in Table 2 to illustrate the usefulness. The analyst only needs to make two modifications by adding "command" and "add move the UAV to ground latitude", to obtain the target synonymous statement.

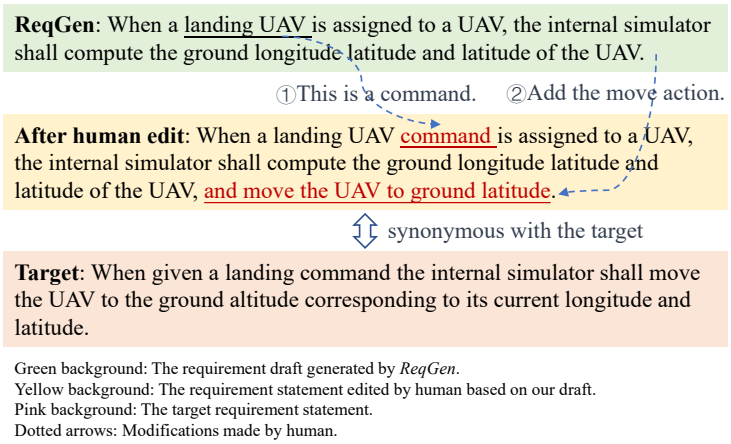


Figure 5. One example showing the usefulness of *ReqGen*.

Second, people may use different terms to represent the same concept, causing ambiguity and difficulty for understandability and automation of follow-up tasks, e.g., traceability. *ReqGen* generates the drafts based on the domain ontology and it can provide domain terms, making the analyst start from the draft, rather than starting from scratch. After human editing, some domain terms still remain. Thus, our approach is helpful for consistent usage of domain terms.

6.3. Limitations

Temporarily, we only randomly selected a few keywords as the seeds of *ReqGen* for the requirements statement generation and did not restrict their roles (e.g., specifying that the subject is mandatory). We plan to explore the impact of syntax role configurations, that the given keywords belong to, on the specifications generation.

Our method cannot replace the elicitation and analysis of users' requirements. In other words, our aim was only to assist engineers to quickly produce the requirements that they already roughly know (i.e., the keywords). Meanwhile, we cannot guarantee that all the specifications are generated with professional domain words and clear formal expressions.

We did not perform an empirical study of the practical usefulness of *ReqGen* in real-world practice. In this initial study, we only evaluated its effectiveness by comparing it with six popular NLG approaches on the common indicators of BLEU, ROUGE, and our proposed RS4RE.

7. Conclusions

This study aimed to automatically generate requirements statements based on predefined keywords. We proposed an approach called *ReqGen*, which fine-tunes UniLM by injecting keywords-related knowledge with repeated emphasis on the most-relevant ones, integrates a copy mechanism to ensure the hard constraint of keyword inclusion, and uses syntax-constrained decoding to cater to syntax requirements. Compared with six popular baselines, we showed that *ReqGen* obtains superior performance on the requirements specification generation task.

Author Contributions: Conceptualization: Z.Z., L.Z. and X.L.; methodology and software: Z.Z., X.G. and H.L.; validation: Z.Z., X.L. and X.G.; formal analysis: Z.Z. and X.L.; resources: Z.Z.; data curation: Z.Z.; writing—original draft: Z.Z.; writing—review and editing: X.L. and L.S.; visualization: Z.Z.; supervision: L.Z. and X.L.; project administration: X.L.; funding acquisition: X.L. and L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the National Science Foundation of China Grant Nos. 62102014 and 62177003. It is also partially supported by the State Key Laboratory of Software Development Environment No. SKLSDE-2021ZX-10.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are available within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Terzakis, J. The impact of requirements on software quality across three product generations. In Proceedings of the 2013 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro, Brazil, 15–19 July 2013; pp. 284–289.
2. Aurum, A.; Wohlin, C. *Engineering and Managing Software Requirements*; Springer: Berlin/Heidelberg, Germany, 2005. [CrossRef]
3. Wiegers, K. *Software Requirements*; Microsoft Press: Redmond, WA, USA, 2013.
4. Lian, X.; Rahimi, M.; Cleland-Huang, J.; Zhang, L.; Ferrai, R.; Smith, M. Mining Requirements Knowledge from Collections of Domain Documents. In Proceedings of the 24th IEEE International Requirements Engineering Conference (RE), Beijing, China, 12–16 September 2016; IEEE Computer Society: New York, NY, USA, 2016; pp. 156–165. [CrossRef]
5. Mavin, A.; Wilkinson, P.; Harwood, A.; Novak, M. Easy Approach to Requirements Syntax (EARS). In Proceedings of the 17th IEEE International Requirements Engineering Conference, Atlanta, GA, USA, 31 August 2009–4 September 2009; pp. 317–322. [CrossRef]
6. Maiden, N.A.M.; Manning, S.; Jones, S.; Greenwood, J. Generating requirements from systems models using patterns: A case study. *Requir. Eng.* **2005**, *10*, 276–288. [CrossRef]
7. Yu, E.S.K.; Bois, P.D.; Dubois, E.; Mylopoulos, J. From Organization Models to System Requirements: A 'Cooperating Agents' Approach. In Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS-95), Vienna, Austria, 9–12 May 1995; pp. 194–204.
8. Letier, E.; van Lamsweerde, A. Deriving operational software specifications from system goals. In Proceedings of the Tenth ACM SIGSOFT Symposium on Foundations of Software Engineering 2002, Charleston, SC, USA, 18–22 November 2002; pp. 119–128.
9. Landtsheer, R.D.; Letier, E.; van Lamsweerde, A. Deriving tabular event-based specifications from goal-oriented requirements models. *Requir. Eng.* **2004**, *9*, 104–120. [CrossRef]
10. Van Lamsweerde, A. Goal-oriented requirements enginering: A roundtrip from research to practice [enginering read engineering]. In Proceedings of the 12th IEEE International Requirements Engineering Conference, Kyoto, Japan, 6–10 September 2004; pp. 4–7.
11. van Lamsweerde, A.; Willemet, L. Inferring Declarative Requirements Specifications from Operational Scenarios. *IEEE Trans. Softw. Eng.* **1998**, *24*, 1089–1114. [CrossRef]
12. Meziane, F.; Athanasakis, N.; Ananiadou, S. Generating Natural Language specifications from UML class diagrams. *Requir. Eng.* **2008**, *13*, 1–18. [CrossRef]
13. Berenbach, B. The Automated Extraction of Requirements from UML Models. In Proceedings of the 11th IEEE International Conference on Requirements Engineering (RE 2003), Monterey Bay, CA, USA, 8–12 September 2003; IEEE Computer Society: New York, NY, USA, 2003; p. 287.
14. Souag, A.; Mazo, R.; Salinesi, C.; Comyn-Wattiau, I. Using the AMAN-DA Method to Generate Security Requirements: A Case Study in the Maritime Domain. *Requir. Eng.* **2018**, *23*, 557–580. [CrossRef]

15. Lian, X.; Cleland-Huang, J.; Zhang, L. Mining Associations Between Quality Concerns and Functional Requirements. In Proceedings of the 25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, 4–8 September 2017; Moreira, A., Araújo, J., Hayes, J., Paech, B., Eds.; IEEE Computer Society: New York, NY, USA, 2017; pp. 292–301. [CrossRef]
16. Lian, X.; Liu, W.; Zhang, L. Assisting engineers extracting requirements on components from domain documents. *Inf. Softw. Technol.* **2020**, *118*, 106196. [CrossRef]
17. Li, Y.; Guzman, E.; Tsiamoura, K.; Schneider, F.; Bruegge, B. Automated Requirements Extraction for Scientific Software. *Procedia Comput. Sci.* **2015**, *51*, 582–591. [CrossRef]
18. Shi, L.; Xing, M.; Li, M.; Wang, Y.; Li, S.; Wang, Q. Detection of Hidden Feature Requests from Massive Chat Messages via Deep Siamese Network. In Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), Seoul, Republic of Korea, 5–11 October 2020; pp. 641–653.
19. Dumitru, H.; Gibiec, M.; Hariri, N.; Cleland-Huang, J.; Mobasher, B.; Castro-Herrera, C.; Mirakhorli, M. On-demand feature recommendations derived from mining public product descriptions. In Proceedings of the 33rd International Conference on Software Engineering, ICSE, Honolulu, HI, USA, 21–28 May 2011; pp. 181–190.
20. Sree-Kumar, A.; Planas, E.; Clarisó, R. Extracting software product line feature models from natural language specifications. In Proceedings of the 22nd International Systems and Software Product Line Conference—Volume 1, SPLC 2018, Gothenburg, Sweden, 10–14 September 2018; ACM: New York, NY, USA, 2018; pp. 43–53.
21. Zhou, J.; Hänninen, K.; Lundqvist, K.; Lu, Y.; Provenzano, L.; Forsberg, K. An environment-driven ontological approach to requirements elicitation for safety-critical systems. In Proceedings of the 23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, 24–28 August 2015; IEEE Computer Society: New York, NY, USA, 2015; pp. 247–251.
22. Khan, J.A.; Xie, Y.; Liu, L.; Wen, L. Analysis of Requirements-Related Arguments in User Forums. In Proceedings of the 27th IEEE International Requirements Engineering Conference, RE 2019, Jeju, Republic of Korea, 23–27 September 2019; pp. 63–74.
23. Astegher, M.; Busetta, P.; Perini, A.; Susi, A. Requirements for Online User Feedback Management in RE Tasks. In Proceedings of the 29th IEEE International Requirements Engineering Conference Workshops, RE 2021 Workshops, Notre Dame, IN, USA, 20–24 September 2021; p. 336.
24. Johann, T.; Stanik, C.; Alizadeh, B.A.M.; Maalej, W. SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In Proceedings of the 25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, 4–8 September 2017; IEEE Computer Society: New York, NY, USA, 2017; pp. 21–30.
25. Dąbrowski, J.; Kifetew, F.M.; Muñante, D.; Letier, E.; Siena, A.; Susi, A. Discovering Requirements through Goal-Driven Process Mining. In Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW), Lisbon, Portugal, 4–8 September 2017; pp. 199–203. [CrossRef]
26. Sleimi, A.; Ceci, M.; Sannier, N.; Sabetzadeh, M.; Bri, L.; Dann, J. A Query System for Extracting Requirements-Related Information from Legal Texts. In Proceedings of the 27th IEEE International Requirements Engineering Conference, RE 2019, Jeju, Republic of Korea, 23–27 September 2019; pp. 319–329.
27. Falkner, A.; Palomares, C.; Franch, X.; Schenner, G.; Aznar, P.; Schoerghuber, A. Identifying Requirements in Requests for Proposal: A Research Preview. In Proceedings of the Requirements Engineering: Foundation for Software Quality—25th International Working Conference, REFSQ 2019, Essen, Germany, 18–21 March 2019; Knauss, E., Goedicke, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11412, pp. 176–182.
28. Devine, P.; Koh, Y.S.; Blincoe, K. Evaluating Unsupervised Text Embeddings on Software User Feedback. In Proceedings of the 29th IEEE International Requirements Engineering Conference Workshops, RE 2021 Workshops, Notre Dame, IN, USA, 20–24 September 2021; pp. 87–95.
29. Henao, P.R.; Fischbach, J.; Spies, D.; Frattini, J.; Vogelsang, A. Transfer Learning for Mining Feature Requests and Bug Reports from Tweets and App Store Reviews. In Proceedings of the 29th IEEE International Requirements Engineering Conference Workshops, RE 2021 Workshops, Notre Dame, IN, USA, 20–24 September 2021; pp. 80–86.
30. Sainani, A.; Anish, P.R.; Joshi, V.; Ghaisas, S. Extracting and Classifying Requirements from Software Engineering Contracts. In Proceedings of the 28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, 31 August–4 September 2020; pp. 147–157.
31. Tizard, J. Requirement Mining in Software Product Forums. In Proceedings of the 27th IEEE International Requirements Engineering Conference, RE 2019, Jeju, Republic of Korea, 23–27 September 2019; pp. 428–433.
32. Dong, L.; Yang, N.; Wang, W.; Wei, F.; Liu, X.; Wang, Y.; Gao, J.; Zhou, M.; Hon, H.W. Unified Language Model Pre-training for Natural Language Understanding and Generation. In Proceedings of the Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, Vancouver, BC, Canada, 8–14 December 2019; pp. 13042–13054.
33. Lin, B.Y.; Zhou, W.; Shen, M.; Zhou, P.; Bhagavatula, C.; Choi, Y.; Ren, X. CommonGen: A Constrained Text Generation Challenge for Generative Commonsense Reasoning. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16–20 November 2020; Findings of ACL; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; Volume EMNLP 2020, pp. 1823–1840.
34. Rothe, S.; Narayan, S.; Severyn, A. Leveraging pre-trained checkpoints for sequence generation tasks. *Trans. Assoc. Comput. Linguist.* **2020**, *8*, 264–280. [CrossRef]

35. Wu, C.; Wu, F.; Qi, T.; Huang, Y. Empowering news recommendation with pre-trained language models. In Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual, 11–15 July 2021; pp. 1652–1656.
36. Guo, W.; Zhang, L.; Lian, X. Putting software requirements under the microscope: Automated extraction of their semantic elements. In Proceedings of the 29th IEEE International Requirements Engineering Conference, RE, Notre Dame, IN, USA, 20–24 September 2021; pp. 416–417.
37. *ISO/IEC/IEEE 29148:2018(E)*; ISO/IEC/IEEE International Standard—Systems and Software Engineering—Life Cycle Processes—Requirements Engineering. IEEE: New York, NY, USA, 2018; pp. 1–104. [CrossRef]
38. Zhao, Z.; Zhang, L.; Lian, X. What can Open Domain Model Tell Us about the Missing Software Requirements: A Preliminary Study. In Proceedings of the 29th IEEE International Requirements Engineering Conference, RE, Notre Dame, IN, USA, 20–24 September 2021; pp. 24–34.
39. Li, Y.; Goel, P.; Rajendra, V.K.; Singh, H.S.; Francis, J.; Ma, K.; Nyberg, E.; Oltramari, A. Lexically-constrained Text Generation through Commonsense Knowledge Extraction and Injection. *arXiv* **2020**, arXiv:2012.10813.
40. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
41. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is All you Need. In Proceedings of the Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 5998–6008.
42. Graves, A.; Mohamed, A.; Hinton, G.E. Speech recognition with deep recurrent neural networks. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, 26–31 May 2013; pp. 6645–6649.
43. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
44. Türetken, O.; Su, O.; Demirörs, O. Automating software requirements generation from business process models. In Proceedings of the 1st Conference on the Principles of Software Eng.(PRISE'04), Buenos Aires, Argentina, 22–26 November 2004.
45. Cox, K.; Phalp, K.; Bleistein, S.J.; Verner, J.M. Deriving requirements from process models via the problem frames approach. *Inf. Softw. Technol.* **2005**, *47*, 319–337. [CrossRef]
46. Osama, M.; Zaki-Ismail, A.; Abdelrazek, M.; Grundy, J.; Ibrahim, A. DBRG: Description-Based Non-Quality Requirements Generator. In Proceedings of the IEEE 29th International Requirements Engineering Conference (RE), Notre Dame, IN, USA, 20–24 September 2021; pp. 424–425.
47. Novgorodov, S.; Elad, G.; Guy, I.; Radinsky, K. Generating Product Descriptions from User Reviews. In Proceedings of the World Wide Web Conference, WWW 2019, San Francisco, CA, USA, 13–17 May 2019; pp. 1354–1364.
48. Novgorodov, S.; Guy, I.; Elad, G.; Radinsky, K. Descriptions from the Customers: Comparative Analysis of Review-based Product Description Generation Methods. *ACM Trans. Internet Technol.* **2020**, *20*, 44:1–44:31. [CrossRef]
49. Elad, G.; Guy, I.; Novgorodov, S.; Kimelfeld, B.; Radinsky, K. Learning to Generate Personalized Product Descriptions. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, 3–7 November 2019; pp. 389–398.
50. Mou, L.; Yan, R.; Li, G.; Zhang, L.; Jin, Z. Backward and forward language modeling for constrained sentence generation. *arXiv* **2015**, arXiv:1512.06612.
51. Liu, D.; Fu, J.; Qu, Q.; Lv, J. BFGAN: Backward and Forward Generative Adversarial Networks for Lexically Constrained Sentence Generation. *IEEE ACM Trans. Audio Speech Lang. Process.* **2019**, *27*, 2350–2361. [CrossRef]
52. Hokamp, C.; Liu, Q. Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Volume 1: Long Papers, Vancouver, BC, Canada, 30 July–4 August 2017; Association for Computational Linguistics: Stroudsburg, PA, USA, 2017; pp. 1535–1546.
53. Miao, N.; Zhou, H.; Mou, L.; Yan, R.; Li, L. CGMH: Constrained Sentence Generation by Metropolis-Hastings Sampling. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 6834–6842. [CrossRef]
54. Sha, L. Gradient-guided Unsupervised Lexically Constrained Text Generation. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, 16–20 November 2020; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 8692–8703.
55. Ding, T.; Pan, S. Personalized Emphasis Framing for Persuasive Message Generation. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, TX, USA, 1–4 November 2016; The Association for Computational Linguistics: Stroudsburg, PA, USA, 2016; pp. 1432–1441.
56. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
57. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
58. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.

59. Keskar, N.S.; McCann, B.; Varshney, L.R.; Xiong, C.; Socher, R. CTRL: A Conditional Transformer Language Model for Controllable Generation. *arXiv* **2019**, arXiv:1909.05858.
60. Du, Z.; Qian, Y.; Liu, X.; Ding, M.; Qiu, J.; Yang, Z.; Tang, J. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, 22–27 May 2022; Association for Computational Linguistics: Stroudsburg, PA, USA, 2022; pp. 320–335.
61. Song, K.; Tan, X.; Qin, T.; Lu, J.; Liu, T. MASS: Masked Sequence to Sequence Pre-training for Language Generation. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, CA, USA, 9–15 June 2019; Volume 97, pp. 5926–5936.
62. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* **2020**, *21*, 140:1–140:67.
63. Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; Zettlemoyer, L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *arXiv* **2019**, arXiv:1910.13461.
64. Bechhofer, S.; Van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D.L.; Patel-Schneider, P.F.; Stein, L.A. OWL web ontology language reference. *W3C Recomm.* **2004**, *10*, 1–53.
65. Gu, J.; Lu, Z.; Li, H.; Li, V.O.K. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, Volume 1: Long Papers, Berlin, Germany, 7–12 August 2016; The Association for Computer Linguistics: Stroudsburg, PA, USA, 2016.
66. Su, J. SPACES: Extract Generate Long Text Summaries. Available online: https://spaces.ac.cn/archives/8046/comment-page-1 (accessed on 1 January 2021).
67. Cleland-Huang, J.; Vierhauser, M.; Bayley, S. Dronology: An incubator for cyber-physical systems research. In Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER), Gothenburg, Sweden, 27 May–3 June 2018; pp. 109–112.
68. *Standard Building Automation System (BAS) Specification*; Technical Report; City of Toronto, Standard Specifications. Available online: https://www.toronto.ca/wp-content/uploads/2017/10/918d-Standard-Building-Automation-System-BAS-Specification-for-City-Buildings-2015.pdf (accessed on 1 October 2017).
69. Manning, C.D.; Surdeanu, M.; Bauer, J.; Finkel, J.R.; Bethard, S.; McClosky, D. The Stanford CoreNLP Natural Language Processing Toolkit. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, Baltimore, MD, USA, 22–27 June 2014; The Association for Computer Linguistics: Stroudsburg, PA, USA, 2014; pp. 55–60.
70. Zhao, Z.; Zhang, L.; Lian, X. A Preliminary Study on the Potential Usefulness of Open Domain Model for Missing Software Requirements Recommendation. *arXiv* **2022**, arXiv:2208.06757.
71. Zhong, H.; Zhang, J.; Wang, Z.; Wan, H.; Chen, Z. Aligning knowledge and text embeddings by entity descriptions. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 17–21 September 2015; pp. 267–272.
72. Zhang, Y.; Wang, G.; Li, C.; Gan, Z.; Brockett, C.; Dolan, B. POINTER: Constrained progressive text generation via insertion-based generative pre-training. *arXiv* **2020**, arXiv:2005.00558.
73. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.J. Bleu: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA, 6–12 July 2002; pp. 311–318.
74. Lin, C.Y. Rouge: A package for automatic evaluation of summaries. In Proceedings of the Text Summarization Branches Out, Barcelona, Spain, 25–26 July 2004; pp. 74–81.
75. Jawahar, G.; Sagot, B.; Seddah, D. What Does BERT Learn about the Structure of Language? In Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Volume 1: Long Papers, Florence, Italy, 28 July–2 August 2019; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 3651–3657.
76. Martín-Lammerding, D.; Astrain, J.J.; Córdoba, A.; Villadangos, J. An ontology-based system to avoid UAS flight conflicts and collisions in dense traffic scenarios. *Expert Syst. Appl.* **2022**, *215*, 119027. [CrossRef]
77. Martın-Lammerding, D.; Astrain, J.J.; Córdoba, A. A Reference Ontology for Collision Avoidance Systems and Accountability. In Proceedings of the Fifteenth International Conference on Advances in Semantic Processing, Barcelona, Spain, 3–7 October 2021.
78. Martín-Lammerding, D.; Astrain, J.J.; Córdoba, A.; Villadangos, J. A Multi-UAS Simulator for High Density Air Traffic Scenarios. In Proceedings of the VEHICULAR 2022, The Eleventh International Conference on Advances in Vehicular Systems, Technologies and Applications, Venice, Italy, 22–26 May 2022; pp. 32–37.
79. Kučera, A.; Pitner, T. Semantic BMS: Allowing usage of building automation data in facility benchmarking. *Adv. Eng. Inform.* **2018**, *35*, 69–84. [CrossRef]