On-Demand Security Requirements Synthesis with Relational Generative Adversarial Networks

Viktoria Koscinski
Rochester Institute of Technology
Rochester, NY, USA
vk2635@rit.edu

Sara Hashemi
Rochester Institute of Technology
Rochester, NY, USA
sxhics@rit.edu

Mehdi Mirakhorli
Rochester Institute of Technology
Rochester, NY, USA
mxmvse@rit.edu

***Abstract*—Security requirements engineering is a manual and error-prone activity that is often neglected due to the knowledge gap between cybersecurity professionals and software requirements engineers. In this paper, we aim to automate the process of recommending and synthesizing security requirements specifications and therefore supporting requirements engineers in soliciting and specifying security requirements. We investigate the use of Relational Generative Adversarial Networks (GANs) in automatically synthesizing security requirements specifications. We evaluate our approach using a real case study of the Court Case Management System (CCMS) developed for the Indiana Supreme Court's Division of State Court Administration. We present an approach based on RelGAN to generate security requirements specifications for the CCMS. We show that RelGAN is practical for synthesizing security requirements specifications as indicated by subject matter experts. Based on this study, we demonstrate promising results for the use of GANs in the software requirements synthesis domain. We also provide a baseline for synthesizing requirements, highlight limitations and weaknesses of RelGAN and define opportunities for further investigations.**

***Index Terms*—Software Security Requirements, Requirements Engineering, Generative Adversarial Networks**

I. Introduction

Software requirements engineering is the process of identifying, analyzing and documenting the necessary attributes, characteristics, or qualities of a system and features expected by the users [1]. It consists of: *requirements elicitation*, or the gathering of requirements for the software; *classification* and *categorization* of the requirements; *analysis* and *validation* to confirm the requirements with stakeholders; requirements *specification* to develop concise and unambiguous write-ups for the software requirements.

In practice, requirements specifications tend to focus on the functional needs of stakeholders, therefore ignoring important security aspects [2] and often failing to capture and specify security requirements [3], [4], [5]. Historically, security has been an afterthought, where systems have been developed first, and then developers have identified and added required security features [6]. Even when security requirements are solicited, they are developed and specified independently of other requirements and requirements engineering activities. As a result, many vital security requirements are repeatedly ignored. Additionally, requirements engineers are not necessarily cybersecurity experts [7], [8], [9], and security experts may not have a high level of knowledge about the system being created. As a result, security requirements are prone to being neglected [10]. These challenges are exacerbated by the fact that the majority of tools for requirements engineering [11], [12], [13], [14] either focus on functional requirements, or if focused on security, are then manual in nature.

The main goal of this paper is to design and develop a technique to automate the process of developing security requirements and generate contextualized security requirements that are effective, complete, clear, and consistent. In this paper, we choose to use a Generative Adversarial Network (GAN) [15], which is a model consisting of a *generator* G and a *discriminator* D. G and D play a two-player minimax game, where G tries to create more realistic security requirements that can fool D, and D tries to differentiate between the synthetic security requirements generated by G and the real data. GANs were generally designed to produce synthetic images. However, in recent years, more work [16], [17], [18], [19], [20] has been dedicated to generating sequential, discrete and text-based synthetic data mimicking real samples to address challenges in imbalanced or small datasets [21], [20], [17]. Due to the nature of GANs and the fact that GANs consist of two machines, the generator can be built on various machine learning-based models, ranging from Convolutional Neural Networks (CNNs) to Long Short-Term Memory (LSTM) and Recurrent Neural Networks (RNNs), based on the application of the research purpose [16], [20], [17], [21].

While GANs have achieved impressive results in producing meaningful and realistic samples in prior studies related to image and text generation [22], [20], their practicality for *synthesizing software requirements* has not been investigated. In particular, this is a challenging task because of the semantics and expressiveness of natural language in comparison to random noise in image data. The novel contribution of this paper lies in investigating whether the practicality of GANs in other application domains is translatable to the software requirements domain. This paper conducts an in-depth case study of using the Relational Generative Adversarial Network (RelGAN) for generating and recommending security requirements specifications for a large and real software system. RelGAN is a type of GAN that uses relational memory and CNNs, which have previously demonstrated useful in generating image captions [20]. We use the Court Case Management System (CCMS), developed for Indiana Supreme Court's

Division of State Court Administration, as a case study. This paper investigated the following research questions:

- **RQ1:** *Can RelGAN be promising for generating security requirements?* We found that RelGAN generates synthetic security requirements similar to the original requirements of the CCMS. RelGAN's performance metrics in terms of diversity of generated samples and matching between the generated samples and original data are comparable to RelGAN's application in other domains. This result highlights that the use of GANs can be a promising approach for synthesizing and recommending requirements.
- **RQ2:** *Does the proposed method result in high quality security requirements specifications?* We found that when the synthesized requirements are grammatically correct, the generated specifications follow the best practices of writing security requirements. Overall, 68% of all synthesized security requirements had no specification defects and out of those which were grammatically correct, 87% had no specification defects.
- **RQ3:** *Are the synthesized security requirements useful in practice?* 9 subject matter requirements engineering experts with prior requirements specification experience evaluated the synthesized security requirements. They indicated that the majority (80%) of syntactically correct requirements were useful (22%) or very useful (58%). However, 42% of the syntactically incorrect requirements were perceived useful despite grammatical errors in the specifications.
- **RQ4:** *Would the adjustment of input requirements based on rules of writing requirements impact the quality of synthesized requirements?* Our experimentation results did not show improvement over the quality of synthesized requirements after such treatment. In fact, the study that preprocessed the requirements to simplify them and improve the writing quality resulted in more ambiguous specifications.

The **contributions** of our work are four-fold:

- An empirically grounded investigation of the generative adversarial method for synthesizing security requirements.
- An automated functional prototype to generate security requirements specifications.
- A real case study and qualitative subject matter expert evaluation to demonstrate the practical significance of the approach, as well as its limitations.
- An in-depth discussion of limitations, weaknesses and failure points of RelGAN requiring further investigation.

To support reproducibility of the findings, all our data, evaluations, source code, pipelines and prototype will be released at github.com. The remainder of this paper is organized as follows: Section II briefly introduces various concepts and terms related to GANs to ensure that the essence of the paper can be understood by a broader audience. Section III discusses the case study design and the methodology followed to conduct this study. Section IV presents the results achieved. Section V discusses the results observed. Section VI contains related works. Section VII elaborates on threats to the validity of this work. Section VIII concludes this paper.

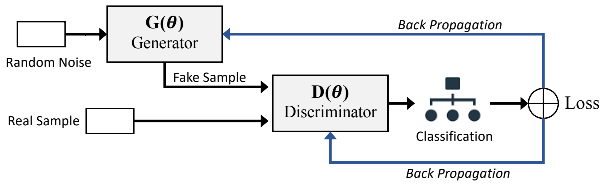

Fig. 1. The architecture of a traditional GAN.

II. Background and Foundation of GANs

<u>G</u>enerative <u>A</u>dversarial <u>N</u>etworks (GANs) [15] are a new emerging neural network architecture. GAN model architecture typically consists of two adversarial models (see Figure 1), a *generator* and *discriminator* that compete against each other. The final goal of these types of machines is to generate synthetic samples (e.g., security requirements) that are indistinguishable from real samples. The generator model captures the data distribution, producing synthetic samples, and the discriminator model learns to determine whether an incoming sample is real or a synthetic version imitating real samples. Learning is conducted through a competition between the generator and discriminator, each improving themselves until synthetic data (security requirements) are indistinguishable from real ones from the perspective of the domain. One of the main challenges in training GANs is considered finding an appropriate balance between both machines and preventing mode collapse. This is due to the dynamic nature and simultaneous training of both machines at the expense of one another in GANs. Mode collapse and inability to converge are a result of imbalanced learning. Pre-training, whether one or both machines, has shown to be effective in this matter [23]. While it is possible to train a language GAN from scratch, leveraging maximum likelihood estimation pre-training can improve the GAN and is among the most successful approaches to text modeling [24], [23].

Studies of GANs in image generation applications have demonstrated that a good generative model can synthesize new examples that are not just plausible, but are indistinguishable from real images. GANs are a rapidly evolving field, known to deliver remarkable results in areas such as synthesizing images from textual descriptions or image classification [17], [20], [21], [22]. However, to the best of our knowledge, there has not been an empirical investigation of using GANs for synthesizing software security requirements, and therefore, supporting requirements engineers.

III. Case study Design

To investigate whether it is possible to automatically synthesize realistic and useful security requirements, we develop an approach using RelGAN [20], a new Generative Adversarial Network (GAN) architecture. The process of synthesizing and recommending security requirements is initiated when a user provides a preliminary set of functional requirements about a product. These functional requirements will then be parsed and analyzed by our approach to create security requirements for the given system. We evaluate this approach on a large and real software system.

A. Case Selection

To answer our research questions from Section I, we conducted an in-depth case study with one industrial case [25] based on guidelines for industrially-based case studies [26]. The unit of analysis in our study is a software project. We use a real requirements document for developing a Court Case Management System (CCMS) for the Indiana Supreme Court's Division of State Court Administration. This document specifies the software requirements for a CCMS to be built for the courts and clerks in the state of Indiana.

This project was selected because (i) it is widely used in the real world by a large user base, (ii) it contains complex functional and non-functional requirements, including security requirements, (iii) it has a total of 223 functional and 137 technical (non-functional) requirements, categorized into subcategories describing what each subset of requirements is about; of the technical requirements, 88 requirements are classified as "security" requirements.

We designed a ***two-stage*** study setup to conduct our investigation, wherein **quantitative metrics** were used to measure the performance of our approach in terms of standard metrics for evaluating GANs, followed by **qualitative analysis** of the generated security requirements to confirm the findings and obtain a deeper understanding in terms of usefulness, practicality and quality of the requirements specifications.

B. Selected GAN Architecture for Security Requirements Generation

Dominant GANs use Long Short-Term Memory (LSTM)-based generators. However, in this paper, we use RelGAN, a type of GAN built on combining the power of a *relational memory*-based generator with multiple embedded representations provided through the *Convolutional Neural Network*-based discriminator. The training is conducted through the Gumbel-Softmax relaxation technique [27], [28], enabling training GANs on discrete data. We make the above choices to address several challenges associated with the usage of LSTM-based GANs for text synthesis. In particular, prior studies have demonstrated that LSTM-based GANs will suffer from severe discriminator loss, mode collapse and poor performance in long text sentence generation [20].

Figure 2 demonstrates an overview of our setup. Our design uses pre-training to avoid mode collapse as a result of imbalanced learning. In pre-training, we train a model on real software requirements samples and save it with its adjusted weights. In particular, we rely upon using maximum likelihood estimation (MLE) in the pre-training phase, detailed in section III-C2. This enables the generator to learn the features prior to adversarial training and elevate its learning based on a better initialization than random noise. Consequently, this setup will provide a more balanced competition between the generator and discriminator, therefore mitigating mode collapse and poor performance and providing a more rapid progress in generative learning.

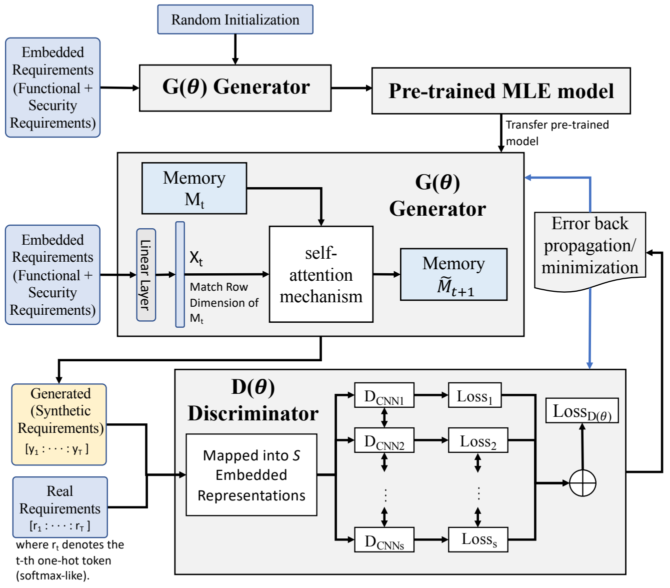
Random Initialization
Embedded Requirements (Functional + Security Requirements)
G(θ) Generator
Pre-trained MLE model
Transfer pre-trained model
Memory M_t
G(θ) Generator
Linear Layer
X_t
Match Row Dimension of M_t
self-attention mechanism
Memory M̃_{t+1}
Error back propagation/ minimization
Embedded Requirements (Functional + Security Requirements)
Generated (Synthetic Requirements) [y_1 : ⋯ : y_T]
Real Requirements [r_1 : ⋯ : r_T]
where r_t denotes the t-th one-hot token (softmax-like).
D(θ) Discriminator
Mapped into S Embedded Representations
D_CNN1
D_CNN2
D_CNNs
Loss_1
Loss_2
Loss_s
Loss_D(θ)

Fig. 2. Synthetic security requirement generation using Relational Generative Adversarial Networks.

In the adversarial training phase, we transfer the pre-trained model and use the pre-trained generator in adversarial training. The pre-trained generator maps a function from a noise distribution to the data space. We use the self-attention mechanism to update the memory from M_t to M_{t+1} by incorporating new observations X_t from real requirements specifications that are fed to the generator. M_t and M_{t+1} represent two consecutive memory slots at time t and $t+1$, respectively. Then, the discriminator provides a probability of whether the incoming sample came from training data or the generator output. Based on the determined loss, the generator and discriminator's parameters are simultaneously updated and trained. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

RelGAN consists of three major components: a relational memory generator for long-distance dependency modeling, Gumbel-Softmax relaxation for training the GAN on discrete data and multiple embedded representations in the discriminator for a more informative signal for updating the generator.

1) Generator: The generator produces synthetic security requirements samples that can be identified as real data. This type of neural network-based machine is first trained with real world requirements to provide a pre-trained model to avoid mode collapse and imbalanced learning between the GAN components. The pre-trained model is used to receive loss output and feedback from the discriminator, leading to a closer match between synthetic and real requirements samples. In this work, the input to the generator (G((θ))) consists of embedded representations of textual requirements. The generator's architecture is based on relational memory. Relational memory utilizes relational information derived from language models and empowers them to generate logical content. To this end, when an embedded input (x) is presented to the generator, it is passed through a *linear layer*, assigning it to a respective row dimension to match the memory matrix M_t, consisting of various memory slots as rows. The *self-attention* mechanism acts by incorporating the X_t matrix and updating the memory matrix M_t by *row-wise* concatenation of M_t and X_t. The result of this mechanism is a new matrix denoted as M_{t+1}. In this regard, the generator's input has been designed to accept a mixture of functional and security text requirements which are concatenated with a memory function.

The generator's goal is to synthesize security requirements based on relational memory. Relational memory considers a memory matrix with the ability of interaction between the memory slots. It uses a *self-attention* mechanism [29] to enable better language modeling. In comparison with LSTM-based machines, it has shown promise in increasing the power of the generator in text generation [20]. The input file is adjusted based on maximum length of all input sentences to provide a fixed length to the relational memory generator. The relational memory module in the generator builds an output based on the requirements' vocabulary size, sequence length and additional hyper-parameters (see RelGAN paper [20]), such as *temperature*, *batch size*, *memory slots* and *memory heads* (see footnote[1] for parameters).

2) Discriminator: The discriminator (D((θ))) accepts real (functional and security) requirements samples in a $d \times T$ format, as input, in addition to the generated requirements obtained through the generator. In real samples, d is the dimension of the embedded vector for each word and T represents the input sentence length. The basis for the discriminator's architecture is an embedded combination of convolutional neural network (CNN) classifiers designed to capture various aspects of the discrete input sentences. CNNs consist of a series of convolutional filters alongside pooling layers, providing insight to various feature levels of the input. Multiple CNN discriminators are connected through weight sharing, leading to multiple losses, which are the basis for the average discriminator loss. The CNN architecture of the discriminator consists of 4 convolution and pooling layers. Each convolution layer consists of 300 filters with 2, 3, 4 and 5 sizes, respectively. Each convolution layer is followed by a max pooling layer; the size of pooling is based on the respective convolution filter size and the pooling layer's input sequence length with [1, 1, 1, 1] strides. The convolution and pooling layers are followed by a dropout layer of 0.75 rate, fully connected layer and linear output layer with the Adam optimizer [30] used in the network architecture.

[1]The following parameters are used: Vocabulary (2029-2066); Sequence (49); Temperature (1000); Batch_size (64); Memory (256); Memory slot (1); Memory head (2)

C. RelGAN training

Our training datasets contain examples from 8 different source documents in 5 different domains, as shown in Table I. We set aside security requirements from the Indiana CCMS document for use in our case study. After obtaining the 1186 requirements, we shuffled them randomly and split them into training (949 requirements) and testing sets (237 requirements). The $\sim$80-20% split is due to access to a limited number of samples and due to the training dataset being used for every part of the experiment besides obtaining metrics (which the testing data is used for). Shuffling requirements randomly before splitting them up helped ensure that testing data is a representative subset of training data, with a similar ratio of security and functional requirements. Data is presented to RelGAN in different sequence (sentence) lengths. Therefore, the sequence length is automatically set to the maximum sentence length in the training data file by RelGAN prior to the pre-training step. Vocabulary size is similarly adjusted. Padding is used to adjust the input as to provide a fixed length of requirement sentences to the generator and discriminator.

1) Preprocessing: We conducted basic preprocessing on this data to ensure that the requirements data was consistent, to reduce possible nonsense outputs from the GAN, and to help us better evaluate our generated requirements. This manual preprocessing step consisted of: removing instances of "e.g." and "for example," changing phrases such as "he or she" into a single pronoun, separating requirements with bullet points into multiple requirements (one per bullet), splitting requirements that contained long lists of actions into multiple requirements, fixing punctuation so that each requirement ends with a period and so that punctuation marks do not contain unnecessary spaces, removing unnecessary descriptions or details inside parentheses (but not removing acronyms in parentheses), and separating requirements that contained multiple sentences into multiple requirements (one requirement per line). While this preprocessing reduced requirements' complexity and length, it did not focus on guidelines for writing quality requirements. In other words, the focus was solely on providing RelGAN with consistency to help generate realistic sentences. Because our preprocessing step included splitting up certain requirements, our number of collected requirements grew to 1186.

2) Pre-training: We use the Maximum Likelihood Estimation (MLE) technique for pre-training the generator. This is done by obtaining the cross-entropy loss with a softmax activation function. The generator is trained with the input requirements and the tokens generated by the model are used to make the next prediction, providing a sequence output. After obtaining the predictions, we are then able to calculate the sum of the negative log likelihoods. We have used security and non-security requirements to introduce context to the system for which we are generating the security requirements. As MLE has a tendency to be biased based on the input data distributions which are different from those in the inference stage, combining security and non-security requirements enables training a more generalized model. We find that pre-

TABLE I
DOMAINS OF REQUIREMENTS CONTAINED IN OUR DATASETS.

Domain	Document/project name	# Sec.	# Funct.
Education	Access 4 Learning Community	115	-
Judicial	Court Case Management System (CCMS) developed for Indiana Supreme Court's Division of State Court Administration	88	301
Judicial	Judicial Council of California Facilities Services: Computer Aided Facilities Management 2.0	73	268
Healthcare	Vermont Health Care Uniform Reporting and Evaluation System 3.0	156	70
HR	Maryland Department of Information Technology Statewide Personnel System	97	-
Generic	NIST Minimum Security Requirements for Multi-User Operating Systems	14	-
Generic	OWASP Application Security Verification Standard 4.0	47	-
Generic	The European Union Agency for Cybersecurity Indispensable baseline security requirements for the procurement of secure ICT products and services	45	-

training the generator using MLE for 30 epochs with a batch-size of 64 and learning rate of 1e-2 using the Adam optimizer before adversarial training provides a good initialization for the generator and eventually prevention of mode collapse. The pre-trained model and weights are saved and used as the base for transferring the model to the generator and fine-tuning the generator in the adversarial training step.

3) Adversarial Training: In the adversarial training step, we transfer the pre-trained model in step (2) and fine-tune the generator based on the saved model instead of using random initialization. The pre-trained model is fed with random noise and generates synthetic requirements, which are then passed along to the discriminator with the real samples. The learning rate is set to 1e-4 for the adversarial training. The Gumbel-Softmax relaxation technique has been used to pass generator loss to the generator and bypass the "non-differentiability" issue when working with discrete data generation. Furthermore, the temperature parameter was set to 1000 for a better approximation of labels, enabling more exploration and generation of diverse samples.

4) Inference: We tested RelGAN on real-world requirements. For this phase we used 237 testing samples which were set aside from the beginning. 46% of our testing examples are security requirements, which is the same ratio as in the 949 training samples. This phase includes the use of the test samples to generate BLEU scores and compare synthetic with never-before-seen real world requirements.

D. Experimentation

Our experimentation simulates a scenario in which a requirements engineer has functional requirements for the Indiana Court Case Management System (CCMS), but not security requirements. For the training data, we use security requirements obtained from 7 out of 8 sources, since we exclude Indiana CCMS security requirements, and we include all the functional requirements.

We conduct quantitative and qualitative evaluations to investigate whether RelGAN is promising in this domain (RQ1); if it can synthesize high quality security requirements from the standards of writing good requirements (RQ2); if the synthesized requirements are useful in practice to help requirements engineers (RQ3); and if adjusting the quality of training data based on the standards of writing good requirements will impact the quality of the synthesized requirements (RQ4). For answering RQ4, we create two versions of our training data. Study #1 uses the original data, and Study #2 uses an enhanced preprocessing to manually improve the requirements' quality using guidelines for writing quality requirements, as described in Section IV-B. Beyond improving the quality of the training data from a GAN standpoint, enhanced augmentation also improves the requirements from the standpoint of requirements engineering best practices [31]. Enhanced preprocessing tasks include replacing weak phrases, removing subjective phrases, and removing implicit subjects. In order for the requirements to be non-ambiguous and complete, some requirements had to be further split, slightly increasing our number of samples.

IV. RESULTS

A. RQ1: Can RelGAN be promising for generating security requirements?

We measure the performance of RelGAN using standard metrics applied to GANs. To measure sample diversity, we generate the *negative log-likelihood* (NLL$_{gen}$) for the synthetic and real data [20], [32]:

$$NLL_{gen} = -\mathbb{E}_{r_{1:T} \sim P_r} \log P_\theta(r_1, \ldots, r_T), \tag{1}$$

where $r_{1:T}$ represents the set of real sentences provided to the GAN, P_r is the real sentence distribution, and P_θ is the generated sentence distribution. A low sample diversity, characterized by a high NLL$_{gen}$ score, is an indicator of *mode collapse*, a situation where GAN generates the same data. Therefore, sample diversity also gives us an idea as to what extent RelGAN is suffering from mode collapse when generating requirements.

To measure *sample quality*, we obtain *Bilingual Evaluation Understudy (BLEU)* scores [33], which assess the similarity between two sentences. BLEU scores are obtained by comparing n-grams of a generated *candidate sentence*, with n-grams of a set of *reference sentences*. Using n-grams, as opposed to single words, helps differentiate realistic sentences from sentences that simply consist of a generated set of "realistic" words in a random order. BLEU scoring also utilizes a *brevity penalty*, which helps ensure that generated sentences are of

TABLE II

NLL$_{gen}$ AND BLEU SCORES OF OUR CASE STUDIES, COMPARING 2, 3, 4, AND 5-GRAMS OF SYNTHESIZED DATA TO REFERENCE DATA.

Study	NLL$_{gen}$	BLEU-2	BLEU-3	BLEU-4	BLEU-5
Study 1 (Original Requirements)	0.2370	0.6208	0.4075	**0.2920**	**0.2154**
Study 2 (Syntacticly Improved Requirements Specification)	**0.1790**	0.6006	0.3826	0.2680	0.2093
RelGAN in other domain (Image Caption Study)	0.655	**0.743**	**0.467**	0.264	0.156

TABLE III

INDIANA COURT CASE MANAGEMENT SYSTEM (CCMS) CASE STUDY: SAMPLE OF 10 ORIGINAL (REAL) SECURITY REQUIREMENTS AND ASSOCIATED RECOMMENDED (SYNTHETIC) SECURITY REQUIREMENTS. THIS DEMONSTRATES THAT NOT ONLY CAN THE APPROACH GENERATE REAL AND USEFUL REQUIREMENTS, BUT ALSO THAT THE QUALITY OF SYNTHESIZED REQUIREMENTS IS COMPARABLE TO THOSE WRITTEN BY HUMAN EXPERTS.

#	Original Security Requirement	#	Recommended Synthetic Security Requirements
R.1	The user interface shall dynamically reflect functions and capabilities that are consistent with the user's rights and privileges.	S.1	The system shall provide the capability to automatically tailor the content presented to every user based on their security profile.
R.2	The system must provide the capability to log security violations.	S.2	The system shall log security events.
R.3	The system must provide the capability for security administrators to define security roles which control user's abilities to access screens, data and perform system functions.	S.3	The system shall provide the capability to administrators to restrict data access by user type.
R.4	The system must provide the capability to ensure screens, functions or menu items are only visible to authorized users.	S.4.1	The system shall provide the capability to automatically tailor the content presented to every user based on their security profile.
		S.4.2	The system shall provide the capability to administrators to restrict data access for each security profile by application screen or page.
R.5	The system must allow authorized users or administrators to search and view the audit log as well as the archived audit logs.	S.5	The system shall protect the security audit trail from unauthorized use.
R.6	Tracking must include but not limited to action date/time, user who performed the action, action taken or changes made, business entities affected by this action, field values before and after changes are made, etc.	S.6.1	The system shall provide the capability to log key attributes of audited events in the audit trails, to include but not limited to the user id of a person performing the action.
		S.6.2	The system shall provide the capability to log key attributes of audited events in the audit trails, to include but not limited to the date and time.
		S.6.3	The system shall provide the capability to log key attributes of audited events in the audit trails, to include but not limited to the event type.
R.7	The system must encrypt passwords and prevent password from being displayed and printed.	S.7	The efm shall store passwords in a one-way encrypted form.
R.8	The system must allow for multiple methods of authentication.	S.8	The system shall require multi-factor authentication.
R.9	The application shall support configurable password strength rules.	S.9	The system shall provide the capability for administrator to define required password characteristics.
R.10	The system must track user actions requiring auditing based on business rules.	S.10	The system shall provide an audit of all files uploaded by court users.

a realistic length, compared to the reference sentences. The brevity penalty BP, is defined by Papineni *et al.* as follows:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r \end{cases}, \tag{2}$$

where c is the length of the candidate sentence, and r is the reference set length. The BLEU score is then defined as:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \tag{3}$$

where N is the maximum size of n-gram being calculated (in our case 5), w_n is a set of positive weights summing to one, and p_n is the geometric average of the modified n-gram precisions (a precision measure that takes into account the maximum number of times a word occurs in a reference sentence). Similar to other works [20], [17], [34], we use our 237-requirement test dataset as the set of reference sentences.

Table II reports the NLL$_{gen}$ and BLEU scores for studies 1 and 2. To better interpret these scores as well as the cross-domain performance of RelGAN, we intentionally include the scores for RelGAN from the original paper [20] that applied the method on image captions (text). The intention is to demonstrate and compare the performance of RelGAN across these two domains (image captions vs software security requirements). This demonstrates whether we can achieve similar performance for requirements synthesis.

RelGAN achieves an NLL$_{gen}$ score of 0.237 in Study 1 and an even lower score of 0.179 in Study 2 that used requirements with improved writing as input. NLL$_{gen}$ evaluates the density of the real data on the generator model. Models with better sample diversity would have a broader coverage over the real data space, and therefore, a lower NLL$_{gen}$ loss. Comparing it with the original RelGAN paper on image captions (NLL$_{gen}$ of 0.655 vs. 0.179 in Study 2), we achieve a higher diversity.

The higher the BLEU score, the better the results and the better the matching has been between the generated samples and the real ones. This matching means that synthetic samples have been able to mimic the behavior of the original real set. The BLEU-2 score for our study is 0.6208, which is close to RelGAN image caption study, with a score of 0.743. We further investigate the practical significance and quality of requirements through RQ2 and RQ3.

Finding #1: The NLL_{gen} and BLEU scores indicate that quantitative evaluation of RelGAN for security requirements synthesis is promising and is in line with other applications of GANs for text synthesis. However, these scores alone do not measure the practical significance of our approach and will not indicate whether the synthesized security requirements are useful in practice.

B. RQ2: Does the proposed method result in high quality security requirements specifications?

To further evaluate the quality and practicality of the synthesized security requirements, we also perform a human evaluation. We recruited 9 subject matter experts (SMEs) who were familiar with software security and had experience writing software requirements specifications, including security requirements. These SMEs were provided with a summary description of the Court Case Management System (CCMS) and were asked to evaluate the *usefulness* and *practicality* of the synthesized security requirements for enhancing the security of the CCMS software. Additionally, each SME was asked to evaluate the quality of generated security requirements based on the criteria of writing good requirements [31]. In particular, the SMEs considered the following relevant metrics:

- **Non-Ambiguity:** The requirement has a unique interpretation.
- **Completeness:** The extent to which all of the parts of the requirement are present, and each part is fully developed.
- **Understandability:** Each requirement is fully understood when used by the user/requirements engineer.
- **Specification quality:** Requirements shall not contain:
  - Implicit Subject Sentences: Requirements in which the subject is not explicitly stated, but is meant to be understood from the context. E.g., "*they* shall be encrypted."
  - Multiple Sentences: Requirements that contain more than one sentence.
  - Optional Sentences: Requirements that leave an option up to the developer. E.g., "the system can *either use passwords or PIN codes*."
  - Subjective Sentences: Requirements containing terms that can be interpreted in multiple ways. E.g., "passwords shall use *many* types of characters."
  - Underspecified Sentences: Requirements lacking important details. E.g., "the system shall use *an* encryption algorithm."
  - Weak phrases: Can, could, might, etc.
- **Usefulness:** Is this requirement useful for identifying relevant security requirements for this system?

Table IV shows results of our quality analysis and our inter-rater reliability (IRR) of the SMEs for 9 unique quality categories. For grammatically/syntactically correct requirements in each defect category, the IRR ranges from .7222 to .9931, demonstrating a minimum error rate for each category. Conflicting answers have been peer-reviewed and resolved. Our study shows that the most common defects of the synthesized requirements are ambiguous sentences and incomplete sentences, with 23% and 16.2% of all requirements from both case studies falling into these categories, respectively. A reason for this is that these categories often encompass others. E.g., if a requirement is not understandable, it is usually also ambiguous. The least common defects were *implicit subject*, *multiple sentences*, and *weak phrase*, which did not apply to any requirements. Because software requirements have an underlying structure, and preprocessing ensured that there was only one sentence per line, the generator had many more good examples to learn from than poor ones, leading to not generating requirements with these defects. It is also important to note that out of all generated *grammatically correct* requirements from both case studies, only 7.7% and 2.6% are ambiguous and incomplete, respectively.

Table III lists 10 sample synthesized requirements and their closest mapping to the original requirements of the CCMS. We have also color-coded the similar concepts in each security requirement. This table illustrates samples of highest quality requirements generated by RelGAN that directly map to original security requirements of the CCMS. It is important to reiterate that CCMS security requirements were eliminated from the training data and only the CCMS's functional requirements were fed to RelGAN to provide context for the synthesis of new security requirements.

Finding #2: RelGAN can synthesize requirements without many defects. Grammatically incorrect requirements are the biggest source of defects within the generated requirements. Eliminating the grammatically incorrect requirements significantly reduces the number of requirements with specification defects (e.g., *ambiguous specifications* reduces by over 15%).

C. RQ3: Are the synthesized security requirements useful in practice?

We evaluated the usefulness of the requirements using two methods. First, we manually reviewed both recommended synthetic requirements and the original security requirements (intentionally omitted from the training data). We mapped each original requirement to an equivalent synthetic requirement. This mapping analysis was performed by two individuals independently and the results were discussed and peer-evaluated together. We found that 26.14% and 29.55% of original security requirements were recommended using our generated synthetic requirements for both studies, respectively.

We evaluated the usefulness of the synthetic requirements by asking our SMEs if they find the generated requirements useful for consideration by the CCMS software. The SMEs evaluated the usefulness of the requirements using the Likert Scale (1 to 5), 5 meaning the requirement is *extremely useful*, and the rest of the scale indicating *very useful*, *somewhat useful*, *not very useful*, and *not at all useful*.

TABLE IV
AN ANALYSIS OF REQUIREMENTS DEFECTS FOR ALL GENERATED SECURITY REQUIREMENTS AND GRAMMATICALLY CORRECT GENERATED SECURITY REQUIREMENTS, ALONG WITH THE INTER-RATER RELIABILITY SCORES FOR GRAMMATICALLY CORRECT REQUIREMENTS.

Defect Category	**% of Security Requirements**						**Inter-Rater Reliability**	
	All Requirements			**Grammatically Correct Only**				
	Study 1	**Study 2**	**Total**	**Study 1**	**Study 2**	**Total**	**Study 1**	**Study 2**
Ambiguous Sentence	25.3	**20.5**	23.0	**03.6**	11.5	07.7	0.7837	0.7514
Incomplete Sentence	21.3	**11.0**	16.2	03.6	**01.6**	02.6	0.8452	0.7978
Not Understandable	18.7	**08.2**	13.5	01.8	**00.0**	00.9	0.8810	0.9299
Implicit Subject	**00.0**	**00.0**	00.0	**00.0**	**00.0**	00.0	0.9311	0.9526
Multiple Sentences	**00.0**	**00.0**	00.0	**00.0**	**00.0**	00.0	0.9782	0.9572
Optional Sentence	**01.3**	01.4	01.4	**00.0**	01.6	00.9	0.9534	0.9781
Subjective	**02.7**	05.5	04.1	**01.8**	06.6	04.3	0.9147	0.9381
Underspecified	**08.0**	09.6	08.8	**03.6**	06.6	05.1	0.7292	0.7222
Weak Phrase	**00.0**	**00.0**	00.0	**00.0**	**00.0**	00.0	0.9931	0.9818
Defect-Free Requirements	68.0	**68.5**	68.2	**87.5**	78.7	82.9	-	-

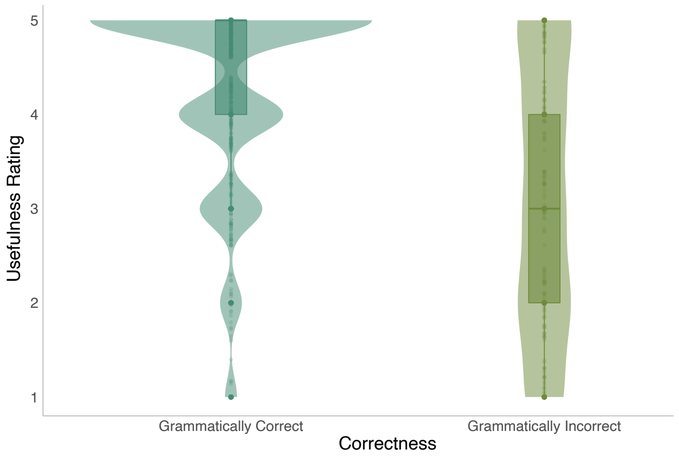

Fig. 3. Violin Plot: SMEs' Evaluation of Usefulness in Study #1.

Figure 3 demonstrates violin plots for the usefulness of the synthetic requirements as evaluated by the SMEs (Study 1). We did not report violin plots for Study 2 since the results were not significantly different from Study 1. The median of usefulness for grammatically correct requirements is 5 (extremely useful), while for grammatically incorrect security requirements it is 3 (somewhat useful). On average, the requirements' usefulness was 3.96 for Study 1 and 3.78 for Study 2. For grammatically correct requirements, average usefulness is 4.28 and 3.95 for studies 1 and 2, respectively.

Finding #3: Grammatically correct security requirements were perceived as extremely useful by the SMEs and practical for the system under study. Furthermore, the direct mapping synthesized high-quality requirements directly to the original system, suggesting the high-quality and usefulness of the generated requirements.

D. Would the adjustment of input requirements based on rules of writing requirements impact the quality of synthesized requirements?

We evaluated the impact of quality of requirements specifications used in the training on the quality of synthesized requirements. Table IV reports the quality metrics across. The results indicate that there was no improvement due to the quality improvement on the data. Requirements with enhanced preprocessing ended up resulting in more ambiguous specifications. More research is needed on how different types of preprocessing affect the results and what preprocessing methods are optimal.

Finding #4: The treatment of training requirements with a preprocessing to enhance their specification quality did not have a significant impact on the specification of synthesized security requirements.

V. DISCUSSIONS, LIMITATIONS, FUTURE WORK

Our results indicate that usage of GANs for generating and recommending security requirements specifications is promising. However, further research is needed to enhance the quality and completeness of such synthetic requirements to better support requirements engineers. Here we provide a deeper discussion of limitations and opportunities for improvement.

A. Syntactically Incorrect Requirements

We found that 21% of generated security requirements across both case studies had syntactic issues. Some of these were minor, but others impact the readability and understandability of the security requirements. The following list provides examples of grammatically incorrect synthesized requirements:

- Synthetic #1: *"Solutions will support the database that is encrypted at the court staff"* instead of "by the court staff."
- Synthetic #2: *"The system shall have the capability to administrator to restrict data access by role, based a clerk filing fee."*
- Synthetic #3: *"The system shall capture the capability to administrators to restrict data access by user type such as but not limited to the submission of test data from a data from data submitters."*

While synthetic requirement #1 only has a minor issue, synthetic requirement #2 appears to be similar to "*The system must* *have central administration capability to add, edit and delete court, government, and no-fee users/staff with appropriate rights management*," but RelGAN has failed to synthesize a syntactically correct requirement with a close meaning.

B. Support of Creativity in Security Requirements Engineering

There has been an increasing number of research papers on creativity and requirements engineering [35]. This research area emphasizes the need for creativity to identify enhancements and unexpected requirements which make a product outperform its competitors. In our case study of the Indiana Court Case Management System, we identified a subset of synthesized security requirements which are considered important for Indiana's CCMS but have been missing in its documentation. Out of the grammatically correct security requirements, 21.43% had a rank of 4 or 5, but did not map to original Indiana CCMS security requirements for Case Study 1. For Case Study 2, this number was 14.75%. For instance, the following synthesized requirements were indicated practical by SMEs but not present in the CCMS.

- Synthetic #4: "*The application shall not reuse a session established prior to authentication.*"
- Synthetic #5: "*The system shall discard data that fails a check for double encoding.*"
- Synthetic #6: "*The system shall provide the capability to generate warning message to notify administrator if multiple security profiles applied to a single user are conflicting.*"
- Synthetic #7: "*The application shall consider anything sent from the browser/client to the server as potentially modified and malicious.*"
- Synthetic #8: "*The system will classify information assets in accordance with security/compliance requirements.*"
- Synthetic #9: "*The system shall provide an audit of all files uploaded by court users.*"

Some of the above security requirements are technical, and closer to code. Synthetic requirement #5, for instance, specifies a mitigation technique for obfuscating attacks using encoding, where the system shall not use data that cannot be verified for double encoding attack. RelGAN has been able to develop new specifications that are not present in the CCMS.

C. Mode Collapse

A common form of GAN failure is called mode collapse. Mode collapse occurs whilst training the GAN, where the generator synthesizes samples that are very similar or identical. In such situation, the generator fails to synthesize data as diverse as the distribution of the real-world data. In each iteration, the generator over-optimizes, resulting in the discriminator's inability to learn and exit the trap. One of the most common outcomes is poor data generation, providing non-diverse samples. We pursued solutions from a data-wise approach and the RelGAN approach. We increased the dataset by combining requirements from multiple sources, initiating a more diverse input affecting the course of training. From the RelGAN point of view, keeping the adversarial training steps low (at 30 steps) and providing a higher temperature (1000 as opposed to 100) increased the generated requirement diversity. A step range of over 50 highly decreased the requirement generation diversity and increased the run-time.

In contrast to image generation, mode collapse in text generation can be measured by directly calculating n-gram statistics. To investigate whether mode collapse occurred or not, we measured mode collapse by the percentage of unique n-grams in the set of generated security requirements, as shown in Table V. Compared to the training dataset, the synthesized security requirements are more unique for bi-grams, 5.8% less unique for tri-grams, and 6.9% less unique for quad-grams. Given the similar diversity, mode collapse does not significantly affect our synthesized requirements.

TABLE V
MODE COLLAPSE: DIVERSITY STATISTICS OF REAL AND SYNTHETIC DATA FROM STUDY #1.

Data	**Unique Bi-grams**	**Unique Tri-grams**	**Unique Quad-grams**
Synthetic Data	51.2%	65.3%	73.2%
Training Data (real)	50.6%	71.1%	80.1%

D. Duplicate Security Requirements

Our analysis demonstrated that RelGAN may generate security requirements which are the duplicate of original requirements in the training data. The more similar security requirements in the training data, the more likely the GAN is to generate a duplicate of one of them. It has occurred, for instance, when we had several requirements related to authentication, audit trail, password strengths, where some were more specific than others. For example, if the training dataset has different versions of a requirement like "the system shall use multi-factor authentication," there are fewer ways the GAN can generate it uniquely. Nevertheless, from the SMEs' perspective, the duplicate requirements were as valuable as high-quality synthesized requirements.

E. Specificity of Synthesized Requirements

One observation is that the majority of the synthesized security requirements are *under-specified* and lack details of the domain data types, actions and assets. While in the case of security requirements this might be acceptable, there are instances of security requirements that would benefit from more specificity. For instance, the following original CCMS requirements are specific to assets, users, actions and entities.

- *"The system must provide functionality to view and print case filings/documents based on user authorization and access rights."*
- *"The system must allow the registrant to register as a Filing User or Registered User, as defined in Proposed Trial Rule 86, on the eFiling system and select a username and password for filing cases online with the court."*
- *"The system must have central administration capability to add, edit and delete*

court, government, and no-fee users/staff with appropriate rights management."

RelGAN has failed to generate substantial requirements with this level of specificity. Further research is needed to investigate whether additional functional and non-functional requirements, sample requirements from the same application domain, or the usage of functional requirements to generate initial noise for the generator would address this problem.

F. Comparison to Human-Written Requirements

In this section, we discuss how the quality of the synthesized requirements compares to human-written requirements. First, Finding #3 indicates that synthesized requirements are considered extremely useful in practice by SMEs. Furthermore, our analysis of directly mapping synthesized requirements to original requirements suggests that synthesized requirements are as good as those written by human experts (Table III). We performed a deeper analysis of the results. One metric we used to assess the quality of the synthesized requirements is BLEU score, measuring similarity of synthesized data to real data based on n-gram analysis. The BLEU-2 (bigram) scores range from 0.60 to 0.62, meaning that synthesized requirements contain bigrams that mimic those of human-written requirements well, without exactly copying the real requirements. While the similarity of the synthesized requirements to handwritten requirements suggests that they are comparable in content, they are still diverse. This is shown by the NLL$_{gen}$ scores, which are 0.18 - 0.24, suggesting good diversity, as well as by the n-gram statistics compared to real data shown in Table V; these results show that the diversity is close to that of handwritten requirements, further suggesting a comparable quality.

The quality of synthesized requirements compared to real written requirements is exemplified in more details in Table III. The left column demonstrates a requirement written by a human SME for a real system (CCMS), and the right column demonstrates an equivalent requirement synthesized by our approach without the knowledge of the requirements in the left column. This table shows similarity not just in the content of requirements, but also in the way they are written. While some synthesized requirements are more concise than human-written requirements (S2 and S8 compared to R2 and R8, respectively), the average sentence length for our human-written training data and our synthesized requirements is very close. Additionally, the majority of our synthesized requirements follow the best practices of using "shall" (74%) or "must" statements (14%), as real requirements do. Only (4%) of synthesized requirements used "should" statements and (6%) used "will" statements. For our training data, a majority also contains "shall" statements (52%) (also "must" (23%), "should" (8%), "will" (14%), "need" (1%), "can," "may," "ought," and "required" (negligible)). Section V-B shows that even more technical synthesized requirements have a quality similar to that of handwritten requirements, and that they can be useful even if they are not directly related to the original handwritten requirements for the system.

Regarding requirements that contain syntactic issues, it can be seen in Section V-A that the quality of syntactically incorrect synthesized requirements ranges from very similar to human requirements that simply have a typo (example 1), to errors that are unlikely to be made by a human (examples 2 and 3). Requirements with more severe syntactic errors are fairly easy to spot and remove, especially with the use of automated techniques such as grammar checking, leaving us with synthesized requirements similar to those that are human-written.

G. Limitations of RelGAN for Requirements Synthesis

One of the limitations of RelGAN is related to the need for tuning the *temperature* parameter [27], [28]. Although the Gumbel-Softmax technique enables working on discrete data by solving the non-differentiable issue, it also causes more sensitivity to the temperature parameter compared to SeqGAN [17] and LeakGAN [18], as they are not reliant on the Gumbel-Softmax technique. Sensitivity to temperature can affect the method and expose it to severe mode collapse or affect sample diversity and quality. We have found the best balance in RelGAN with a temperature between 100-1000.

VI. Related Work

A. Requirements Elicitation and Generation

Requirements generation or elicitation techniques have been developed to aid in the manual process of creating quality requirements that meet stakeholder demands. While many techniques focus on manually or automatically generating formal models *from* requirements [11], [36], [37], [38], [39], [40], [41], [42] to aid in tasks such as requirements analysis or software development, others generate requirements from formal models of a system. For example, techniques have been created to automatically generate requirements statements from *i** models [43], [44], UML models and class diagrams [45], [46], [47], and xtUML diagrams [48]. One technique automatically generates scenarios from Use Case Correspondence Models [49], while another automatically generates formal Object Constraint Language specifications into natural language [50]. While such techniques are useful for systems containing these types of models, not all models have the same type of requirements elicitation support. Our proposed approach aims to solve this issue by providing requirements elicitation without generating a formal model in a specific language for a system.

Other requirements generation techniques are manual or semi-automated, requiring user input or validation steps. The Security Quality Requirements Engineering Methodology [51] consists of 9 comprehensive steps and is manual, providing options for potential tool support. Another approach provides automated conformance checking to requirements templates [52], giving requirements engineers information they need to manually generate higher-quality requirements. Semi-automated techniques include: eliciting security requirements with misuse cases [53], goal-oriented requirements engineering with anti-models [54], a requirement reuse-based approach leveraging use cases and misuse cases [55], deriving requirements from process models using problem frames [56], an automatic requirements generator based on Business Process Re-engineering theory [57], and a series of model transformations and concept mappings to generate security and privacy cloud service requirements from highly abstract organizational goals, requiring some user input throughout its steps [58]. Some of these techniques suffer from the same issue as the formal model-based techniques, since they also require specific technical approaches, while all of the techniques require a substantial amount of manual input, requiring additional time and expertise within the security requirements domain.

The Framework for Eliciting Security Requirements project [59] elicits security requirements for a system, given the functional requirements, similar to our main task. This framework consists of detecting the system's weaknesses through a question template based on functional requirements, obtaining the system's operational environment, obtaining the security objectives of the system based on mappings of the weaknesses, and generating the security requirements. This technique is limited by the small number of weaknesses manually analyzed and by the need of requirement analysts to answer the question template to identify security assumptions and policies. Our proposed technique offers more automation and is not limited based on mapped weaknesses. The reason we chose to use a GAN technique [20] to generate security requirements was due to the promising results GANs have recently had in text generation. To our knowledge, there currently do not exist techniques to generate security requirements using GANs.

VII. Verifiability and Threats to Validity

In our work, we choose some commonly used metrics for the purpose of evaluating RelGAN. Although quantitative metrics, such as NLL_{gen} and BLEU scores give us an idea of how RelGAN performs, this type of analysis alone is insufficient in measuring the practical significance of our approach. There are two main components to measuring the practical significance: requirement quality and relevance. To measure both of these components, we provide an extensive qualitative analysis of our results, including 9 subject matter experts (SMEs) in a review of the quality and relevance of the requirements. We used a real case study for investigating our research questions. We carried out a qualitative study and aimed for practical significance rather than statistical significance. While the results were promising for the system under the study and perceived valuable by requirements engineering SMEs, we do not claim generalizability across all systems. Further research is needed to investigate the generalizability of the approach to other application domains.

Because manual analysis is prone to bias, we have taken steps to mitigate this threat. We recruited SMEs who are familiar with software security and requirements engineering. This helped ensure that our evaluators understood the specification quality guidelines provided to them, and that they were able to accurately judge how useful requirements would be to a Court Case Management System (provided a summarized description of its functions). Additionally, we calculated the inter-rater reliability (IRR) of our extensive quality analysis for grammatically correct requirements to help ensure that the error in our final results is low. The IRRs ranged from .9931 to .7222, showing minimal error. Including grammatically incorrect requirements, the IRRs range from .6526 to .9848. These IRRs are lower because grammatically incorrect requirements are often not fully understandable, incomplete and can be interpreted in multiple ways, which is why we separately report results for grammatically correct requirements.

Another threat to validity is that the results of any GAN reflect its inputs. Because of this, our generated requirements' quality depends on that of the input requirements. It is possible that there are errors in the real input data due to the human nature of writing requirements. We mitigate this threat by choosing input requirements from companies, state governments, or security guidelines documents, which are likely to be written with care by an expert, reducing the likelihood of poorly written requirements. Additionally, being able to input requirements from real systems has the benefit of adding new data (e.g., applying this work to a new domain), or removing any requirements that may be hindering performance.

VIII. Conclusions

We investigate the use of Generative Adversarial Networks (GANs), in particular RelGAN, in synthesizing security requirements specifications. GANs have demonstrated successful results in generation of images, or captions. Our case study demonstrates positive results and highlights areas of weaknesses for synthesizing useful security requirements. While our case studies only focused on the judicial domain, we believe that our use of GANs for generating and recommending security requirements can be generalized to other domains. In fact, because our input data contains requirements from multiple domains, such as education and healthcare, we find that RelGAN generated some requirements relating to these domains as well, which suggests that with the right input examples, the GAN can be guided to generate security requirements for many domains. User evaluation of the requirements indicates the ability of RelGAN in synthesizing real and practical security requirements specifications. In particular, the grammatically correct specifications were significantly valuable for requirements engineer SMEs, with median ranking of 5 (extremely useful). Further research is needed to improve the quality of synthesized requirements to address the problem of underspecification. Our experimentation limited the scope of the RelGAN feasibility study to deriving security requirements based on the functional requirements. It is anticipated that inclusion of other non-functional requirements or constraints may provide additional context. However, excluding such requirements from our experiments will not invalidate the promising results of the paper. Additional study is required to investigate the performance of RelGAN in light of other forms of requirements. Furthermore, we have not investigated generation of non-functional security requirements.

Acknowledgement

This work was partially funded by the US National Science Foundation under grant number CCF-1943300.

References

[1] B. Berenbach, D. Paulish, J. Kazmeier, and A. Rudorfer, *Software amp; Systems Requirements Engineering: In Practice*, 1st ed. USA: McGraw-Hill, Inc., 2009.

[2] M. T. J. Ansari, D. Pandey, and M. Alenezi, "Store: Security threat oriented requirements engineering methodology," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 2, pp. 191–203, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157818306876

[3] D. Ameller, C. Ayala, J. Cabot, and X. Franch, "Non-functional requirements in architectural decision making," *IEEE Software*, vol. 30, no. 2, pp. 61–67, 2013.

[4] L. Chen, M. Ali Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *IEEE Software*, vol. 30, no. 2, pp. 38–45, 2013.

[5] L. Cysneiros and J. do Prado Leite, "Nonfunctional requirements: from elicitation to conceptual models," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 328–350, 2004.

[6] J. Steinmann and O. Ochoa, "Supporting security requirements engineering through the development of the secure development ontology," in *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, 2022, pp. 151–158.

[7] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens, "Supporting requirements engineers in recognising security issues," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2011, pp. 4–18.

[8] M. Hilbrich and M. Frank, "Enforcing security and privacy via a cooperation of security experts and software engineers: a model-based vision," in *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*. IEEE, 2017, pp. 237–240.

[9] M. Bruckschen, C. Northfleet, D. Silva, P. Bridi, R. Granada, R. Vieira, P. Rao, and T. Sander, "Named entity recognition in the legal domain for ontology population," in *Workshop Programme*. Citeseer, 2010, p. 16.

[10] D. M. Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetrò, T. Conte, M. T. Christiansson, D. Greer, C. Lassenius, T. Männistö, M. Nayabi, M. Oivo, B. Penzenstadler, D. Pfahl, R. Prikladnicki, G. Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, J. L. de la Vara, and R. Wieringa, "Naming the pain in requirements engineering," *Empirical Software Engineering*, vol. 22, no. 5, pp. 2298–2338, 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9451-7

[11] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 14, no. 3, pp. 277–330, 2005.

[12] P. Sawyer, P. Rayson, and K. Cosh, "Shallow knowledge as an aid to deep understanding in early phase requirements engineering," *IEEE Transactions on Software Engineering*, vol. 31, no. 11, pp. 969–981, 2005.

[13] J. Badger, D. Throop, and C. Claunch, "Vared: verification and analysis of requirements and early designs," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014, pp. 325–326.

[14] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014, pp. 183–192.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[16] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin, "Adversarial feature matching for text generation," in *International Conference on Machine Learning*. PMLR, 2017, pp. 4006–4015.

[17] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[18] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang, "Long text generation via adversarial training with leaked information," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[19] K. Lin, D. Li, X. He, Z. Zhang, and M.-T. Sun, "Adversarial ranking for language generation," *Advances in neural information processing systems*, vol. 30, 2017.

[20] W. Nie, N. Narodytska, and A. Patel, "Relgan: Relational generative adversarial networks for text generation," in *International conference on learning representations*, 2018.

[21] R. Shu, T. Xia, L. Williams, and T. Menzies, "Dazzle: Using optimized generative adversarial networks to address security data class imbalance issue," in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2022, pp. 144–155. [Online]. Available: https://doi.ieeecomputersociety.org/10.1145/3524842.3528437

[22] W. Fedus, I. Goodfellow, and A. M. Dai, "Maskgan: better text generation via filling in the_," *arXiv preprint arXiv:1801.07736*, 2018.

[23] C. de Masson d'Autume, S. Mohamed, M. Rosca, and J. Rae, "Training language gans from scratch," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[24] T. Che, Y. Li, R. Zhang, R. D. Hjelm, W. Li, Y. Song, and Y. Bengio, "Maximum-likelihood augmented discrete generative adversarial networks," *CoRR*, vol. abs/1702.07983, 2017. [Online]. Available: http://arxiv.org/abs/1702.07983

[25] P. Runeson and M. Hoest, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, 2009.

[26] J. Verner, J. Sampson, V. Tosic, N. A. A. Bakar, and B. Kitchenham, "Guidelines for industrially-based multiple case studies in software engineering," in *Third IEEE International Conference on Research Challenges in Information Science*, 2009, pp. 313–324.

[27] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[28] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

[29] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International conference on machine learning*. PMLR, 2019, pp. 7354–7363.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] "Ieee recommended practice for software requirements specifications," *IEEE Std 830-1998*, pp. 1–40, 1998.

[32] S. Lu, L. Yu, S. Feng, Y. Zhu, and W. Zhang, "Cot: Cooperative training for generative modeling of discrete data," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4164–4172.

[33] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[34] Y. Zhu, S. Lu, L. Zheng, J. Guo, W. Zhang, J. Wang, and Y. T. Yu, "A benchmarking platform for text generation models. arxiv 2018," *arXiv preprint arXiv:1802.01886*.

[35] J. a. Lemos, C. Alves, L. Duboc, and G. N. Rodrigues, "A systematic mapping study on creativity in requirements engineering," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1083–1088. [Online]. Available: https://doi.org/10.1145/2245276.2231945

[36] M. G. Hinchey, J. L. Rash, and C. A. Rouff, "Requirements to design to code: Towards a fully formal approach to automatic code generation," Tech. Rep., 2005.

[37] D. K. Deeptimahanti and R. Sanyal, "Semi-automatic generation of uml models from natural language requirements," in *Proceedings of the 4th India Software Engineering Conference*, 2011, pp. 165–174.

[38] W. Ben Abdessalem Karaa, Z. Ben Azzouz, A. Singh, N. Dey, A. S. Ashour, and H. Ben Ghazala, "Automatic builder of class diagram (abcd): an application of uml generation from functional requirements," *Software: Practice and Experience*, vol. 46, no. 11, pp. 1443–1458, 2016.

[39] K. Kolthoff, "Automatic generation of graphical user interface prototypes from unrestricted natural language requirements," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1234–1237.

[40] H. Harmain and R. Gaizauskas, "Cm-builder: A natural language-based case tool for object-oriented analysis," *Automated Software Engineering*, vol. 10, no. 2, pp. 157–181, 2003.

[41] V. Ambriola and V. Gervasi, "On the systematic analysis of natural language requirements with circe," *Automated Software Engineering*, vol. 13, no. 1, pp. 107–167, 2006.

[42] M. Ilieva and O. Ormandjieva, "Models derived from automatically analyzed textual user requirements," in *Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*. IEEE, 2006, pp. 13–21.

[43] N. A. Maiden, S. Manning, S. Jones, and J. Greenwood, "Generating requirements from systems models using patterns: a case study," *Requirements Engineering*, vol. 10, no. 4, pp. 276–288, 2005.

[44] N. Maiden, S. Jones, C. Ncube, and J. Lockerbie, "Using i* in requirements projects: Some experiences and lessons learned," *Social Modeling for Requirements Engineering. The MIT Press, Cambridge*, 2010.

[45] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating natural language specifications from uml class diagrams," *Requirements Engineering*, vol. 13, no. 1, pp. 1–18, 2008.

[46] B. Berenbach, "The automated extraction of requirements from uml models," in *Proceedings. 11th IEEE International Requirements Engineering Conference, 2003.* IEEE Computer Society, 2003, pp. 287–287.

[47] B. A. Berenbach, "Comparison of uml and text based requirements engineering," in *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 2004, pp. 247–252.

[48] H. Burden and R. Heldal, "Natural language generation from class diagrams," in *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation*, 2011, pp. 1–8.

[49] K. Goto, S. Ogata, J. Shirogane, T. Nakatani, and Y. Fukazawa, "Support of scenario creation by generating event lists from conceptual models," in *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. IEEE, 2015, pp. 376–383.

[50] D. A. Burke and K. Johannisson, "Translating formal software specifications to natural language," in *International Conference on Logical Aspects of Computational Linguistics*. Springer, 2005, pp. 51–66.

[51] N. R. Mead and T. Stehney, "Security quality requirements engineering (square) methodology," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–7, 2005.

[52] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated checking of conformance to requirements templates using natural language processing," *IEEE Transactions on Software Engineering*, vol. 41, no. 10, pp. 944–968, 2015.

[53] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements engineering*, vol. 10, no. 1, pp. 34–44, 2005.

[54] A. Van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," in *Proceedings. 26th International Conference on Software Engineering*. IEEE, 2004, pp. 148–157.

[55] G. Sindre, D. G. Firesmith, and A. L. Opdahl, "A reuse-based approach to determining security requirements," in *REFSQ*, vol. 3. Citeseer, 2003, pp. 127–136.

[56] K. Cox, K. T. Phalp, S. J. Bleistein, and J. M. Verner, "Deriving requirements from process models via the problem frames approach," *Information and Software Technology*, vol. 47, no. 5, pp. 319–337, 2005.

[57] Z.-B. Gan, D.-W. Wei, J.-L. Zhang, and V. Varadharajan, "Business-process-oriented software requirements automatic generator," in *Third International Conference on Information Technology and Applications (ICITA'05)*, vol. 1. IEEE, 2005, pp. 95–98.

[58] N. Argyropoulos, S. Shei, C. Kalloniatis, H. Mouratidis, A. Delaney, A. Fish, and S. Gritzalis, "A semi-automatic approach for eliciting cloud security and privacy requirements," in *Proceedings of the 50th hawaii international conference on system sciences*, 2017.

[59] H. Li, X. Li, J. Hao, G. Xu, Z. Feng, and X. Xie, "Fesr: A framework for eliciting security requirements based on integration of common criteria and weakness detection formal model," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2017, pp. 352–363.