

# Unlocking Hardware Security Assurance: The Potential of LLMs

Xingyu Meng\*, Amisha Srivastava\*, Ayush Arunachalam\*, Avik Ray†,  
Pedro Henrique Silva‡, Rafail Psiakis‡, Yiorgos Makris\* and Kanad Basu\*

\*University of Texas at Dallas, †Amazon Alexa, ‡Technology Innovation Institute

**Abstract**—System-on-Chips (SoCs) form the crux of modern computing systems. SoCs enable high-level integration through the utilization of multiple Intellectual Property (IP) cores. However, the integration of multiple IP cores also presents unique challenges owing to their inherent vulnerabilities, thereby compromising the security of the entire system. Hence, it is imperative to perform hardware security validation to address these concerns. The efficiency of this validation procedure is contingent on the quality of the SoC security properties provided. However, generating security properties with traditional approaches often requires expert intervention and is limited to a few IPs, thereby resulting in a time-consuming and non-robust process. To address this issue, we, for the first time, propose a novel and automated Natural Language Processing (NLP)-based Security Property Generator (NSPG). Specifically, our approach utilizes hardware documentation in order to propose the first hardware security-specific language model, HS-BERT, for extracting security properties dedicated to hardware design. To evaluate our proposed technique, we trained the HS-BERT model using sentences from RISC-V, OpenRISC, MIPS, OpenSPARC, and OpenTitan SoC documentation. When assessed on five untrained OpenTitan hardware IP documents, NSPG was able to extract 326 security properties from 1723 sentences. This, in turn, aided in identifying eight security bugs in the OpenTitan SoC design presented in the hardware hacking competition, Hack@DAC 2022.

## I. INTRODUCTION

Modern computing systems are built on System-on-Chips (SoCs), as they offer a high level of integration through the use of multiple Intellectual Property (IP) cores [24]. However, this also presents new security challenges, since vulnerabilities in one IP core may affect the security of the entire system [21]. While software and firmware patches can address many hardware security vulnerabilities, some cannot be fixed and require extensive security assurance during the design process. Hence, hardware security validation is imperative to ensure the security and trustworthiness of the design. MITRE’s Common Weakness Enumeration (CWE) for hardware categorizes commonly encountered security weaknesses in hardware designs, including issues with security flow, privilege and access control, reset control, memory and storage, peripherals and on-chip fabric, and debugging and test [1]. Commercial verification tools, such as JasperGold Security Path Verification, Mentor Questa Secure Check, and Tortuga Logic Radix, have been proposed for SoC security verification [8], [10], [20]. However, the effectiveness of these tools depends on the quality of the security properties. Therefore, these properties

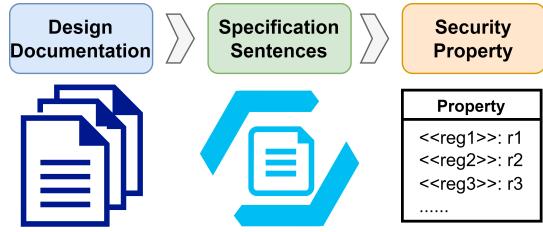


Fig. 1: Security property generation from documents.

are critical components that provide resources for detecting vulnerabilities during SoC development.

Generating appropriate security properties for each specific design can be a challenge. Traditional approaches, which often require the experience of developers, are time-intensive and non-robust [21]. Hence, a technique that is capable of systematically generating security properties for SoCs is needed to address this issue. On the other hand, major organizations such as RISC-V, MSP, and Arduino usually provide documentation describing processor functionalities and operation behaviors with explicit details [7], [9]. Therefore, as shown in Figure 1, we reason that it is possible to generate numerous security properties by analyzing operation details from these documents, which can be converted into security constraints at the Register-Transfer Level (RTL). We aim to achieve this by creating a language-based machine learning framework to extract essential information from the documentation.

Existing research in the biomedical field, such as BioBERT, has shown that text and documents can provide essential information in specialized domains to fine-tune the general Bidirectional Encoder Representations from Transformers (BERT) language model for text generation and phrase classification [15], [35]. Similarly, security property-related sentences in hardware documentation also furnish domain-specific terminology. Hence, we intend to apply this concept to the hardware security domain and develop an automatic security property extraction framework built on the BERT model, which is fine-tuned with domain-specific data, as shown in Figure 2. Our aim is to utilize this framework to extract each sentence in the documentation that can be potentially converted into security properties. To this end, we developed a fully automatic hardware security property generator called NLP-based Security Property Generator (NSPG). NSPG applies data augmentation and masked language model to enhance

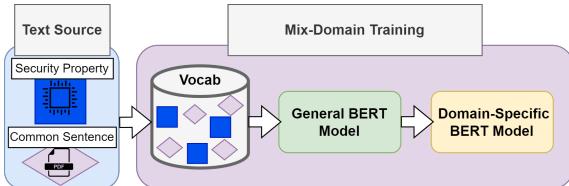


Fig. 2: Domain-specific BERT model.

the dataset and improve the learning process. We compared the output of NSPG with the expected information flow policies of these designs and found that the generated specifications were accurate and comprehensive, and if followed, would protect the designs from known and potential future attacks. To the best of our knowledge, NSPG is the first property generation technique utilizing the NLP model and SoC documentation to generate security properties for hardware designs. Our contributions are summarized as follows:

- 1) We propose an NLP-based security property generation framework, NSPG, to automatically mine security property-related sentences from the SoC documents, assisting in the detection of security vulnerabilities in RTL.
- 2) We present a complete end-to-end framework with hardware domain-specific knowledge and data modification techniques to improve the performance of the proposed HS-BERT model by analyzing hardware documentation.
- 3) NSPG is evaluated on five unseen OpenTitan design documents and all generated security properties are validated. Furthermore, we apply these properties to search for security violations in the OpenTitan design used in Hack@DAC 2022 and identify eight bugs [3].
- 4) When compared with ChatGPT, NSPG shows 15% improvement on identifying security properties in OpenTitan SoC documentation [4].

The rest of this paper is organized as follows. Section II introduces the background of hardware security, and techniques in the NLP domain. Section III includes the proposed technique. Section IV introduces the SoC architecture, threat model, and security objectives used in this work. Section V demonstrates the evaluation of the proposed technique. Section VI discusses the capabilities of the proposed technique. Section VII presents the related works on security property generation. Finally, Section VIII concludes our paper.

## II. BACKGROUND

In this section, we will provide some background on hardware security verification, NLP, and the SoC designs used for developing the proposed NSPG framework.

### A. Hardware Security

1) *Hardware Security Verification*: A plethora of techniques have been developed to ensure the security of software applications, either through source code or binary operations [21]. However, the availability of commercial Electronic Design Automation (EDA) tools, which are specifically crafted for hardware security, is rare. Hence, recently, researchers have focused on developing tools and methodologies to ensure

hardware security. Nevertheless, all existing security verification techniques need robust security properties to validate the trustworthiness and robustness of the RTL [21]. More details on hardware security verification approaches will be presented in Section VII-A.

2) *Hardware Design Documentation*: In order to enhance the comprehensibility of the SoCs, it is imperative to have access to thorough SoC documentation, considering their complexity and dependence on reusable components. However, documentation is generally considered to provide limited value for any scientific analysis, and, thus overlooked by researchers. In reality, technical documentation is an important part of the overall product and should be prioritized along with the design and implementation stages. In more than one instance, insufficient documentation has been regarded as the major reason for design failure [16]. Additionally, various tools have been introduced to generate system documentation more efficiently. For instance, “Javadoc” and “CppDoc” generate API documentation in HTML from the comments in the source code [2], [5].

3) *Hardware Security Property*: The specification of security properties usually varies from the method of security analysis, producing specifications that are unique to the selected verification tools or models [38].

**Listing 1: Security Property and SystemVerilog Assertion**

```

Security Property Description:
If the AES unit wants to finish encryption/decryption
of a data block but the previous output data has not
yet been read by the processor, AES unit is stalled.

SystemVerilog Assertion:
assert property (
  (posedge clk) disable iff (rst)// Security Property
  aes.done |> aes.out == $past(aes.key)
)
else
  // Error Message
  $error("%m previous key has not been read");

```

Listing 1 presents an example of SystemVerilog assertion based on the description of the security property. It shows that in order to generate an appropriate term for a verification mechanism, the descriptions of security properties must contain strong reasoning and details of the operation. Listing 2 shows a paragraph from the OpenTitan AES document [4]. The sentences marked in blue are security related.

**Listing 2: A example paragraph in AES Document**

```

Also, there is a back-pressure mechanism for the
output data. If the AES unit wants to finish the
encryption/decryption of a data block but the previous
output data has not yet been read by the processor,
the AES unit is stalled. It hangs and does not drop
data. The order in which the output registers are read
does not matter. Every output register must be read
at least once for the AES unit to continue. This is
the default behavior. It can be disabled by setting
the MANUAL_OPERATION bit in CTRL_SHADOWED to 1.

```

It is obvious that the security properties have more distinguished contexts, such as the usage of relation conjunctions and domain-specific terms, compared to other sentences in the paragraph. Therefore, it is possible to distinguish the security property-related sentences, which present precise definitions

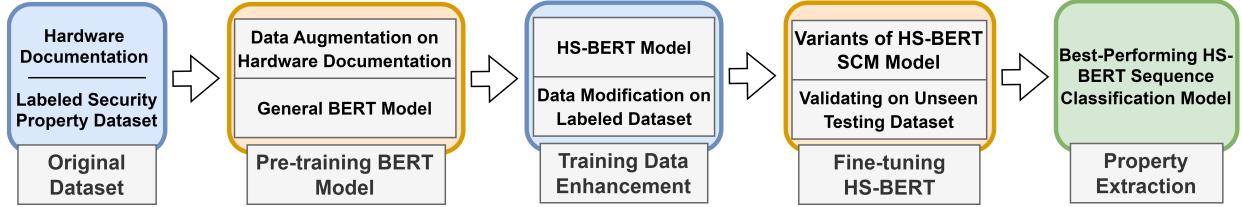


Fig. 3: Proposed NSPG framework.

of the schematic, keywords of the processes, or relations between several entities. Hence, we can emphasize these aspects when fine-tuning the language model by enhancing and identifying the security property-related context in each sentence. Further details about the security property extraction techniques utilized in our framework will be presented in Section III.

#### B. Natural Language Processing

NLP is a field of study that focuses on applying computational techniques to understand, learn, and generate human language content. NLP plays a crucial role in various industries and has a wide range of applications, ranging from real-time translation and social media search engines to sentiment analysis [26]. In this section, we will discuss the NLP facets that are integrated into NSPG.

**1) BERT Model:** Most state-of-the-art natural language models are built on transformer architectures, such as Bidirectional Encoder Representations from Transformers (BERT), which are effective at modeling long-range dependencies in text [27]. These models utilize a multi-layer, multi-head self-attention mechanism, and contextual embeddings which allow for efficient parallel computation on GPUs [23]. **We utilize the BERT masked language model to comprehend the hardware documentation and BERT sequence classification model to extract the security property-related sentences.**

**2) Data Augmentation:** Data augmentation (DA) is a method of enhancing the diversity of training data without collecting more data. It involves adding modified copies of existing data or creating synthetic data to act as a regularizer and reducing overfitting during the training of machine learning models [26]. **Since this is the first work that utilizes design documentation for hardware security, our data samples are limited to open-source documentation. Thus, we will apply DA approaches, as discussed in Section III.**

**3) Masked Language Modeling:** Masked Language Modeling (MLM) is a self-supervised learning method used in state-of-the-art BERT models, such as SciBERT or RoBERTa [15], [23]. MLM is used to improve the ability to comprehend the context and relationships between each word in a sequence, such as the security property demonstrated in Listing 1. **We will utilize MLM to modify the sequences from the document and provide additional data for fine-tuning.**

**4) Sequence Classification Modeling:** Sequence Classification Modeling (SCM) divides sequences into predefined sentiment categories [23]. **Our proposed framework, NSPG, will apply this model to categorize each sequence of**

**sentences and identify if they belong to security property or non-property descriptions in the documents.**

### III. CAN LEVERAGING LARGE LANGUAGE MODELS FOSTER HARDWARE SECURITY ASSURANCE?

In this section, we will demonstrate the details of each process flow in the proposed NSPG framework, as shown in Figure 3. The first step involves comprehending the context of the hardware domain from the hardware design documentation of OpenTitan and RISC-V, and generating the hardware security-specific BERT model. Next, we alter them using data augmentation and hardware domain-specific modification to create an enhanced dataset for fine-tuning the pre-trained BERT model. We compare the performance of various modified datasets used to fine-tune the classification model and select the best one for security property extraction. The details of the data augmentation and pre-training hardware-domain BERT will be discussed in detail in Section III-B. The domain-specific data modification process and security property classification will be described in detail in Section III-C.

#### A. Hardware Documentation Dataset

The sentences used for training and fine-tuning are extracted from various documentation and paragraphs similar to the example shown in Listing 2. We create three datasets, as follows: (1) 15583 sentences from OpenTitan, RISC-V, OpenRISC, MIPS, and OpenSPARC documentation are used for pre-training the BERT model with MLM. This dataset will be called  $\mathcal{D}_{pre}$ . (2) To fine-tune the classification model, we manually labeled training samples consisting of 4427 sentences (out of which 2191 are security property-related sentences and the rest are non-property-related sentences) from the top module and test cases documentation in OpenTitan design, MIPS, OpenSPARC as well as non-privileged documentation for RISC-V. This dataset will be called  $\mathcal{D}_{cls}$ . (3) In order to simulate the scenario of processing unseen documentation, we also manually labeled additional 708 security property-related and non-property-related sentences to validate the fine-tuned SCM model. It consists of 470 security properties and non-properties from OpenTitan AES and ADC documentation, 78 sentences from OpenRISC, and 160 sentences from RISC-V trace documentation. All of these sentences are not included in  $\mathcal{D}_{pre}$ , and this dataset will be called  $\mathcal{D}_{val}$ .

#### B. Comprehending the Hardware Domain

First, the language model needs to recognize the context differences between security properties and regular sentences. General BERT model has been applied in various industries

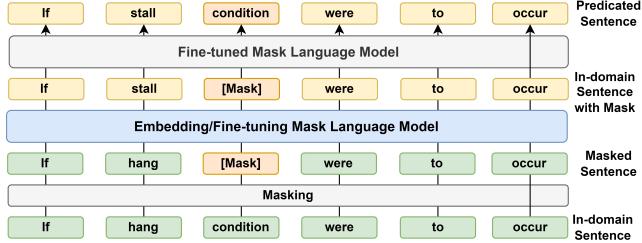


Fig. 4: Masked language model.

and has a wide range of applications, ranging from real-time translation and social media search engines to sentiment analysis [26]. Training the BERT model with its corresponding MLM could augment the performance of the language model by introducing contextual embeddings of the specific domain. As shown in Figure 4, during training, a certain percentage of tokens are randomly selected and replaced with a special token ([MASK] in Masked sentence). After training, the fine-tuned MLM will predict the original tokens based on the context provided by the remaining unmasked tokens, thus completing the in-domain sentence with appropriate phrases. The primary objective is to minimize the cross-entropy loss between the predicted tokens and their original counterparts. For instance, in SciBERT and RoBERTa, by default, 15% of the input tokens are chosen for masking, with 80% probability of being replaced by [MASK], 10% left unchanged and 10% randomly replaced by another token from the vocabulary [15], [23]. Therefore, pre-training BERT model with hardware domain documentation will help it learn to represent the in-domain words based on the context of the other words in the sentence. However, since we have limited data samples for hardware domain-specific documentation (compared to 1.14M papers from Semantic Scholar to train Sci-Bert), data augmentation is needed to improve the dataset [15].

#### 1) Data Augmentation for Hardware Documentation:

Many DA techniques have been proposed for NLP, including rule-based manipulations and generative approaches. Ideal DA techniques should be implemented seamlessly in order to further improve model performance. Techniques that use trained models are more expensive to implement. However, they introduce more variance in data, thereby resulting in enhanced model performance. Although model-based techniques are designed to boost performance in downstream tasks, they are difficult to develop and use. This may result in overfitting or performance degradation when trained on out-of-domain

examples. DA is a key component for enhancing the quantity of in-domain data samples, which involves generating two correlated views of a data point in order to increase the amount and diversity of training data. Although it is important to ensure the diversity of the generated data, the structure, and synonym of the original sentence should not be modified.

Most of the security properties involve the behaviors of two or more entities in the design. Therefore, the context of operational relations and hardware terminology needs to be preserved in fine-tuning MLM. We need to avoid the essential information of the operating entities and only apply DA to swap or replace the rest of the tokens in the sentence. We analyze four DA techniques: random swap, random deletion, synonym replacement, and random insertion. These are the most common data augmentation technique available for NLP model [26]. The closest sentiment texts are provided by Wordnet, a large lexical database of English, where synonyms are interconnected by means of conceptual-semantic, to generate the augmented data [40]. The tokens in the texts are modified in the following ways to generate the augmented data. Table I presents examples of the techniques applied to a sentence from the DA task, as follows:

**Random Swap (RS):** We exchange the positions of two randomly selected phrases such as nouns and conjunctions, while maintaining the operation behavior and the content of the original sentence. For example, as evident from Table I, RS takes parts of the sentence conjunction “when in this mode” and places them into a random spot, which is either before or at end of a conjunction in the original sentence. In case the sentence does not have multiple conjunctions, no augmented sentence will be generated.

**Random Deletion (RD):** To keep the essential context, we only delete one of the adjectives, determiners, or adverbs available in the sentence, which does not impact the operation descriptions. For example, RD removes the definition term “some” from the sentence in Table I. When these terms are not found in the sentence, RD will not be applied.

**Synonym Replacement (SR):** In order to preserve the hardware components described in the sentence, we only replace verbs in the sentence with their closest synonyms obtained from the WordNet database. It utilizes the database to find the closest semantic words for the sentence and replace the verb with the new ones. For example, in Table I, “occur” is replaced with its synonym “happen”, and the sentence still presents the same operation. When the sentence only contains verbs such

TABLE I: Data Augmentation for sentence in documents. The first row shows the original sentence in the document, the rest shows an example sentence after the random swap, random deletion, synonym replacement, and random insertion.

Sentence	
<b>Original</b>	If some hang condition were to occur when in this mode, the main state machine debug register should be read.
<b>Original with RS</b>	If some hang condition were to occur, the main state machine debug register should be read <u>when in this mode</u> .
<b>Original with RD</b>	If <u>{some}</u> hang condition were to occur when in this mode, the main state machine debug register should be read.
<b>Original with SR</b>	If some hang condition were to <u>happen</u> when in this mode, the main state machine debug register should be read.
<b>Original with RI</b>	If some hang condition were to occur when in this mode, the main state machine debug register should be read <u>[immediately]</u> .

TABLE II: The performances of HS-BERT models with the original sentence from  $\mathcal{D}_{pre}$  and after the random swap, random deletion, synonym replacement, and random insertion.

Dataset	Training Samples	Runtime	Perplexity
Original Documents	12472	57 mins	6.018
Original & RS	24065	109 mins	5.018
Original & RD	24946	114 mins	5.046
Original & SR	24208	110 mins	5.029
Original & RI	24931	113 mins	4.795

as “is” or “are”, no augmented sentence will be generated.

**Random Insertion (RI):** For this operation, we first summarize all the adverbs that are used in the documentation, and randomly select one to insert into the sentence near verbs. To avoid altering the context, we randomly choose the most common adverb used in the documents. For instance, insert the adverb ‘immediately’ before or after a randomly selected verb in the sentence, as shown in Table I.

Each method tends to introduce diversity in the original sentence and increase possible word usage, thereby reducing data overfitting and boosting the generalization ability of the model. We train four BERT models using different augmented data (in combination with the original data) and evaluate their performance to determine the best model.

2) *Pre-training BERT with MLM:* The purpose of MLM is to understand the detail of each sentence demonstrated in the document and predict the masked content. Hence, we will select the model having the lowest *perplexity*, where a lower score indicates that the model has a better comprehension of the hardware domain and prediction of the masked word in the in-domain sentences [41]. We split the samples from  $\mathcal{D}_{pre}$  into training and validation datasets with a ratio of 80% and 20%. Each model is trained with samples from the OpenTitan, RISC-V, OpenRISC, MIPS, and OpenSPARC documentation, which consist of 12473 sentences and additional data from the DA task. Table II shows the runtime, and perplexity for each DA approach, as described in Section III-B1. Note that the runtime differences are caused by different numbers of augmented data samples since the operation details of original sentences need to be preserved. We evaluate each pre-trained BERT model with the validation dataset consisting of 3112 sentences. As shown in Table II, the DA task significantly improves the performance of the BERT model comprehension (lower perplexity) with scalable runtime increase. Random insertion operation performs the best among all four approaches with a perplexity score of 5. Therefore, we use the BERT model pre-trained with data from RI for further processes. This BERT model will be called Hardware Security-specific BERT (**HS-BERT**).

### C. Security Property Classification

In this section, we will explain the process of security property classification from the design documentation utilizing the Sequence Classification Model (SCM), as mentioned in Section II-B4. Figure 5 illustrates the training procedure of

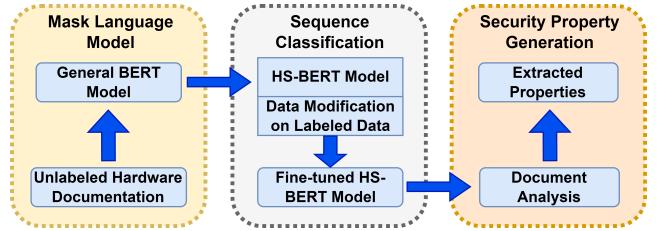


Fig. 5: HS-BERT model training process.

NSPG framework, which consists of three stages. In the first stage, we obtain the HS-BERT model with sentences from hardware design documentation. In the second stage, we will modify the labeled  $\mathcal{D}_{cls}$  with the HS-BERT model to fine-tune SCM. In the third stage, the HS-BERT model will be compared against other state-of-the-art BERT models. Finally, we will validate each fine-tuned SCM with  $\mathcal{D}_{val}$  and utilize the best-performing one to generate security properties from the documents.

$\mathcal{D}_{cls}$  is used to fine-tune the SCM model and validate the performances of each classification model. We will utilize  $\mathcal{D}_{val}$  as unseen in-domain documentation, and evaluate the performances of the fine-tuned SCM model for security property generation. Each BERT sequence classification model is trained with different combinations of the original and modified training datasets to evaluate the classification performance based on accuracy, recall, precision, and F1-score.

Since our priority is to maximize the generation of security properties, we can tolerate the presence of a few non-property-related sentences (*i.e.*, False Positives), which can be removed with further analysis. Therefore, accuracy and recall are the key metrics to determine the performance of sentence modification and SCM in NSPG. Higher values of accuracy and recall indicate better performance of the modification method and SCM model. The details will be discussed in Section III-C4. In this work, we employ the SciBERT and general BERT model (“bert-base-uncased”), and compare their performances against HS-BERT in Section III-C3.

1) *Data Modification for Property Classification:* Data pre-processing and data cleansing are utilized to generate unbiased data, which in turn is critical to furnish reliable machine learning algorithms [25]. Studies have shown that biased learning can result from training on imbalanced or noisy data [34], [46]. Therefore, significant consideration should be provided to the data cleansing process and detailing the methods used in our studies. BERT model tokenizer is used to tokenize the sentences by its data pre-processing technique [23]. In order to improve the performance of sequence classification, we propose a hardware domain-specific modification to further differentiate the features of each sentence in the documents, which enforces the BERT model to recognize more contextual dependencies. For instance, sentences including hardware operations and behavior are considered in-domain, while common descriptions are regarded as out-of-domain.

Although a plethora of data pre-processing approaches have been proposed to improve the performances of machine learn-

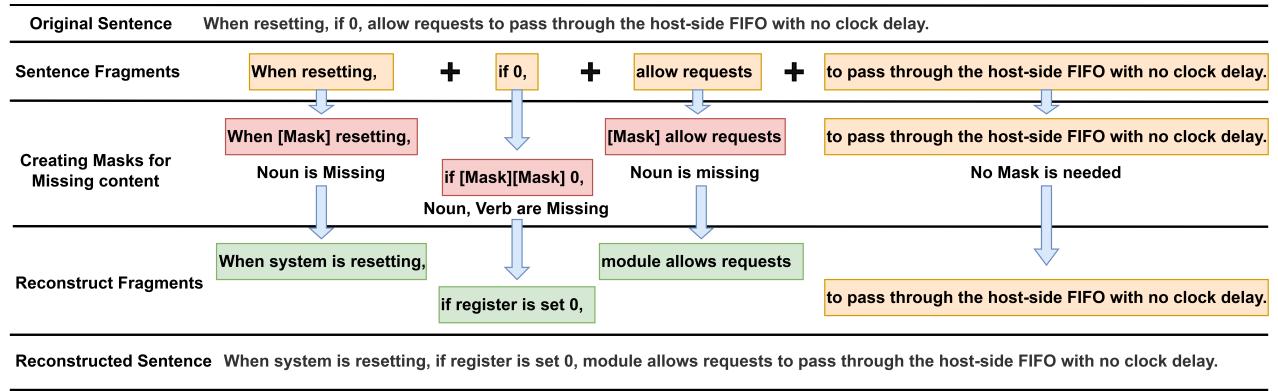


Fig. 6: Sentence modification in documents. The first row represents the original sentence in the document. The second row shows each fragment of the original sentence. The third row adds [Mask] token into the fragment for missing components. The fourth row demonstrates the process of reconstructing the missing components of each fragment, and the last row shows the sentence after constructing the missing components.

ing techniques in text mining and image classification [28], [32], most language models are pre-trained with a wide range of documentation in a different domain. In this work, we consider some data modification methods with hardware security-domain context and validate their impact on model performances. As shown in Table III, we apply several alternative methods, including verb swapping, noun swapping, fragment deletion, and fragment insertion, to preprocess all sentences in the dataset, described as follows:

**Verb Swapping:** This approach substitutes the verbs in the sentence with semantic pre-defined hardware domain-related verbs. For example, in Table III, “stay” and “begins” in each sentence are replaced with “remain” and “starts” to formalize the usage of verbs.

**Noun Swapping:** This method intends to remove the arbitrary term used in each sentence, such as “KEYMGR” and “KEYMGR CTRL STATUS” in Table III, and replace them with more generalized terminology in the hardware domain such as “condition” and “register”.

**Fragment Deletion:** Some fragments of the sentence are not directly related to operation details. In this case, we intend to break the sentence into multiple conjunctions and remove the small fragments such as “begin again” in Table III, while retaining the more essential content of hardware operations.

**Fragment Insertion:** This approach adds information to the incomplete sentence. Typically, we consider a complete sentence consists of two (noun, verb) or three components (noun, verb, and noun). For each incomplete fragment in the sentence,

we will construct them with a similar structure.

2) *Fragment Insertion Modification:* Through our initial analysis, the other data modification approaches furnished sub-par classification performance of up to 55% accuracy, we have included and described the results pertaining to the fragment insertion technique. Therefore, we determine that only fragment insertion improves the performance of the classification model. While noun swapping and verb swapping do not significantly affect performance, fragment removal degrades performance. The primary reason is that fragment insertion adds more in-domain context to the original sentence, which helps the SCM identify the operation context. Therefore, we can infer that instead of simplifying the data, the NLP model tends to require more information and complete structure in each sentence to learn the subjects more accurately. Intuitively, the prevailing assumption is that using more in-domain text in the training datasets should help with domain-specific classification. Although each sentence in the document is constructed differently, we will explore this concept by adding domain-specific portions for each sentence.

Figure 6 shows an example of modifying a sentence in the document with fragment insertion. First, we break the sentence “When resetting, if 0, allow requests to pass through the host-side FIFO with no clock delay.” into fragments by identifying each conjunction “When”, “if”, and “to” in the sentence, and separating the sentence into fragments as: “When resetting”, “if 0”, “allow requests”, and “to pass through the host-side FIFO with no clock delay”. In these

TABLE III: Sentence pre-processing methods and examples.

Sentence	
<b>Original</b>	If KEYMGR is at the conclusion of the operation, KEYMGR CTRL STATUS stays in the same state, begins again.
<b>Verb Swapping</b>	If KEYMGR is at the conclusion of the operation, KEYMGR CTRL STATUS [remain] in the same state, [starts] again.
<b>Noun Swapping</b>	If [condition] is at the conclusion of the operation, [register] stays in the same state, begins again.
<b>Fragment Removal</b>	If KEYMGR is at the conclusion of the operation, KEYMGR CTRL STATUS stays in the same state. {begins again.}
<b>Fragment Addition</b>	If KEYMGR is at the conclusion of the operation, KEYMGR CTRL STATUS stays in the same state, [system sampling] begins again.

TABLE IV: Performance comparison between general BERT, SciBERT and HS-BERT with data modifications from  $\mathcal{D}_{cls}$ .

General BERT				SciBERT				HS-BERT				
	Accuracy	Recall	Precision		Accuracy	Recall	Precision		Accuracy	Recall	Precision	F1-Score
<b>Baseline</b>	83.1%	81%	49.4%	65.1%	83.2%	98%	46%	61.7%	84%	97%	45%	63%
<b>MT</b>	83.2%	93.5%	51.8%	65.8%	82.2%	96%	49%	65%	85.1%	97.3%	48.7%	64.7%
<b>MOT</b>	86%	96.4%	46.7%	63%	88.1%	98%	45%	62%	90.1%	98.3%	49%	65%
<b>MTT</b>	81.2%	83.4%	50.4%	61.2%	87.1%	95%	44%	60%	88.1%	97.5%	48%	64%
<b>MOTMT</b>	83.2%	70.6%	35.5%	71%	87.5%	97%	44%	59%	88.5%	98.1%	48%	64%

fragments, “When resetting”, and “allow requests” are missing nouns, “if 0” is missing noun and verb, and “to pass through the host-side FIFO with no clock delay” is complete. We will add these missing components into the fragments by adding a [Mask] token into the fragment and applying the pre-trained HS-BERT to place appropriate in-domain terms for each [Mask] token. Hence, “When resetting”, “if 0”, and “allow requests” will be transformed into “When system is resetting”, “if value is set 0”, and “module allows requests”. Since the other data modification approaches furnished subpar classification performance of up to 55% accuracy, we have included and described the results pertaining to the fragment insertion technique.

3) *General BERT vs SciBERT vs HS-BERT*: In this section, we will discuss the available pre-trained BERT models and motivate our choice of the proposed HS-BERT, as shown in Section III-B2, for its inclusion in NSPG. General BERT utilizes WordPiece [49] for unsupervised tokenization of input sequences, building its vocabulary with the most frequently used words or sub-word units. SciBERT, on the other hand, is constructed with a new WordPiece vocabulary on a scientific corpus using the SentencePiece1 library [45]. The token overlap between BERT and SciBERT vocabulary is 42%, indicating a substantial difference in the frequently used words between scientific and general domain texts. SciBERT was trained on a dataset comprising 1.14M papers from Semantic Scholar, where 82% of the papers belong to the biomedical domain, while the remaining 18% pertain to computer science [12].

Table IV shows the comparison between the general BERT model, SciBERT, and the proposed HS-BERT, training the same data modification method and the same 3927 samples from  $\mathcal{D}_{cls}$ . Another 500 samples from  $\mathcal{D}_{cls}$  are used for validation. Modifications are applied to both in-domain and out-of-domain sentences. The baseline refers to no modification on both training and testing data. We have considered the following cases for modification: **MT** refers to only modifying training data. **MTT** refers to modifying training and testing data. Note that the modification only changes the content of testing sentences and does not increase the size of test samples. **MOT** refers to both modified and original training data. **MOTMT** refers to the original training data, its modified counterpart, and modified testing data. The results indicate that all three BERT models demonstrate improved performance in comparison to their baseline models, with an average increase of 0.6% accuracy with General BERT, 3.1% accuracy with

TABLE V: Sequence Classification for  $\mathcal{D}_{val}$ . Each row represents the accuracy, recall, and F1-score for different HS-BERT models and different modified labeled datasets.

	OpenTitan		RISCV		Openisc	
	Accuracy	Recall	Accuracy	Recall	Accuracy	Recall
<b>Base-HS-BERT</b>	70.6%	72.8%	71.9%	71.2%	80.3%	80.2%
<b>MT-HS-BERT</b>	74.5%	80.4%	75.8%	79.2%	83.6%	82.6%
<b>MOT-HS-BERT</b>	81.5%	93%	79.1%	90.1%	88.3%	87%
<b>MTT-HS-BERT</b>	75%	85%	74.3%	83.5%	86.3%	82.6%
<b>MOTMT-HS-BERT</b>	76%	84.1%	73.3%	80.5%	87%	84.7%

SciBERT, and 3.95% accuracy with HS-BERT. For each data modification method, the largest effects of fine-tuning were observed in the MOT (+6.1% accuracy with HS-BERT, +4.9% accuracy with SciBERT, and +2.9% accuracy with General BERT) and MOTOT (+4.9% accuracy with SciBERT and +0.1% accuracy with General BERT). While little effect was seen on recall, precision, and F1-score for HS-BERT and SciBERT, General BERT gains a significant improvement in recall when applying MOT (+15.4%). However, HS-BERT, with fine-tuning, outperforms the state of art General BERT and SciBERT model on accuracy and recall, while performing similarly on precision and F1-score. Based on these observations, we determine that HS-BERT is more suitable for extracting potential security properties, which we have subsequently incorporated in NSPG.

4) *Fine-tuning SCM model*: In this stage, NSPG will apply fine-tuned HS-BERT SCM to identify the security properties in the SoC documentation. In order to emulate this scenario in which the SCM is applied on an unseen document to determine whether the sentence is a security property or not, we will test each trained sequence classification model on  $\mathcal{D}_{val}$ . Table V shows the results for HS-BERT performance with no modification and the data modification approach, as described in Section III-C3. **Base-HS-BERT** refers to the SCM performance fine-tuned with no data modification. The HS-BERT model trained with the MOT modification performs the best with an average of 82% accuracy, and 90% recall score. Since this method consists of both the original and the modified sentences, it improves the diversity of the training dataset, which helps the model to learn more features and context of a property-related sentence. Therefore, we will use MOT modification as our final data modification approach for the SCM model.

---

**Algorithm 1** Sentence Formalization

---

**Input:** Document Files, Labeled Dataset  
**Output:** Property, Non-Property

```

1: MLM.train(Document Files)
2: Initialize Enhanced Labeled Dataset
3: for each Sentence in Labeled Dataset do
4:   Split(Sentence) → Fragments
5:   for each Fragment in Fragments do
6:     if Fragment is incomplete then
7:       Apply Data Modification → Fragment
8:     end if
9:   end for
10:  Join(Fragments) → Sentence
11:  Append Enhanced Labeled Dataset ← Sentence
12: end for
13: SCM.train(Enhanced Labeled Dataset)
14: for each Sentence in Document do
15:   if SC.predict(Sentence) > 0.5 then
16:     Sentence → Property
17:   else
18:     Sentence → Non-Property
19: end for
```

---

5) *NSPG Framework Summary*: After validating each procedure in training, we will construct the pipeline of our proposed framework NSPG. It comprises data augmentation, data modification, fine-tuning masked language model, and sequence classification model. The fine-tuned sequence classification model is used to analyze the unseen documents, and extract the security properties which contain essential information about operation behaviors, and register interactions. Algorithm 1 demonstrates the details of each stage for the training and security property generation process. These extracted security properties can be further utilized by commercial hardware verification approaches, such as Cadence Jaspergold, and detect potential vulnerabilities in the design. The generated security properties as well as their application in detecting SoC vulnerabilities will be evaluated in Section V.

#### IV. OPENTITAN SOC AND THREAT MODEL

##### A. OpenTitan SoC

Since we do not have access to commercial designs, our framework has been developed using open-source SoCs, such as OpenSPARC, MIPS, OpenRISC, RISC-V and OpenTitan. In this work, while the OpenTitan design documentation is utilized for both training and validation, the other documents are only used for training. The OpenTitan document considered in our study corresponds to a buggy design, which was used in the Hack@DAC 2022 hardware hacking competition [3].

##### B. Threat Model

Our threat model is similar to the one used in Hack@DAC 2022. The attack scenarios considered are as follows:

- An unprivileged software adversary that can access the core in user mode, with control over user-space interfaces

TABLE VI: Number of processed sentences, security properties generated, and properties covered and not covered by design verification (DV) documentation.

Hardware IP	Sentences	Extracted	Properties	Covered by DV	Not covered by DV
<b>Key Manager</b>	241	51	47	40	7
<b>LC Controller</b>	375	79	76	65	11
<b>HMAC</b>	170	28	27	19	8
<b>KMAC</b>	367	84	74	60	14
<b>OTP Controller</b>	570	105	102	84	18
<b>Total</b>	1723	347	326	268	58

and the ability to issue unprivileged system instructions and request feedback.

- A privileged software adversary that can execute malicious code with supervisor privilege but may target higher privilege levels or bypass security countermeasures.
- An authorized debug adversary which is capable of unlocking and debugging production devices.

The aim of the adversary is to exploit any potential vulnerabilities in the SoC in order to bypass any security objective.

##### C. Security Objectives and Features

The security objectives of OpenTitan are as follows [3]:

- **SO1**: Preventing privilege level compromise due to unprivileged code running in the core.
- **SO2**: Protecting system debug interface from malicious or unauthorized debugger.
- **SO3**: Protecting device integrity and preventing exploits from a software adversary.

Various security features have been developed for the OpenTitan SoC to support the aforementioned security objectives, including privileged access control to peripherals, write and read locks on registers to protect the privilege integrity from unprivileged instruction, and reset to flush sensitive information in cryptographic processors.

## V. EXPERIMENTS

In this section, we will demonstrate the evaluation of our framework NSPG with five OpenTitan IP documents, and discuss its performance on security property extraction. Furthermore, these extracted properties will be utilized with verification methods to detect vulnerabilities in the design.

##### A. Experimental Setup

All of our experiments are run on a server consisting of 40 CPUs of 64-bit Intel(R) Xeon(R) E5-2698 v4 @ 2.20GHz. Our NSPG framework is implemented in Python. We intend to release our framework in GitHub soon, for use by other researchers. The entire SoC documentation comprises of 33 IP design specifications, with 10865 sentences. Each IP design documentation demonstrates information on register descriptions, functionalities, and operation processes, including the security features for various modules, that will be used for evaluating NSPG [7]. Among these, the contents of the operational processes or behaviors can be transformed into security properties, while the others are treated as non-properties. In the following subsections, we will evaluate each extracted

TABLE VII: Examples of security properties generated from the framework.

Module	Constructed Security Properties	HW CWE Category
Key Manager	Upon Disabled entry, the internal key is updated with KMAC computed random values; however, previously generated sideload key slots and software key slots are preserved.	Improper Zeroization of Hardware Register - (1239)
	Invalid state is entered whenever key manager is deactivated through the life cycle connection or when an operation encounters a fault .	Improper Finite State Machines (FSMs) in Hardware Logic - (1245)
	When an illegal operation is supplied, the err_code is updated and the operation is flagged as done with error.	Improper Protection for Outbound Error Messages and Alert Signals - (1320)
	When the life cycle controller deactivates the key manager, the key manager transitions to the Invalid state.	Improper Finite State Machines (FSMs) in Hardware Logic - (1245)
LC Controller	fatal_bus_integ_error_q is triggered when a fatal TL-UL bus integrity fault is detected.	Improper Protection for Outbound Error Messages and Alert Signals - (1320)
	fatal_bus_integ_error_q is set to 1 if a fatal bus integrity fault is detected.	Improper Protection for Outbound Error Messages and Alert Signals - (1320)
HMAC	If SW wants to convert the message byte order, SW should set CFG.endian_swap to 1.	Expected Behavior Violation - (440)
	When the SHA engine is disabled the digest is cleared.	Sensitive Information in Resource Not Removed Before Reuse - (226)
	If CPU writes value into the register, the value is used to randomize internal variables such as secret key, internal state machine, or hash value.	Sensitive Information in Resource Not Removed Before Reuse - (226)
KMAC	If the EnMasking parameter is not set, the second share is always zero.	Improper Zeroization of Hardware Register - (1239)
	If EnMasking is not defined, the KMAC merges the shared key to the unmasked form before uses the key.	Sensitive Information in Resource Not Removed Before Reuse - (226)
	If the EnMasking parameter is set and CFG_SHADOWED.msg_mask is enabled, the message is masked upon loading into the Keccak core using the internal entropy generator.	Security Primitives and Cryptography Issues - (1205)
OTP Controller	The otp_lc_data_o.secrets_valid signal is a multibit valid signal that is set to lc_ctrl_pkg::On if the SECRET2 partition containing the root keys has been locked with a digest.	Improper Prevention of Lock Bit Modification - (1231)
	Read transactions through the CSR window will error out if they are out of bounds, or if read access is locked.	Improper Access Control for Register Interface - (1262)

sentence from five unseen IP documents and identify whether they can be transformed into security properties. Moreover, we will utilize these security properties to search for potential violations in relevant hardware IPs. Since the list of registers in SoC design is available to us, we craft the detailed security properties from the IP security specification and checklist of the OpenTitan SoC registers. The verification of SoC is based on the threat model and security objectives, as described in Section IV-B and IV-C, respectively.

### B. NSPG Framework Evaluation

First, NSPG processes each text file consisting of sentences in the design documentation. It filters out any sentence having less than 10 words, since they usually do not contain enough information about operation behaviors. The rest of the sentences will be parsed through the trained HS-BERT sequence classification model discussed in Section III, and the extracted sentences from each IP document will be listed in property text files. As shown in Table VI, 1723 sentences are processed, and 344 sentences are extracted as potential security properties. Overall, 326 sentences (94% of the 347 extracted sentences) can be utilized to generate security properties for design validation. We compare the generated security properties with the test cases listed in the Design Verification (DV) documentation, which describes all the test cases and IP operations needed to be checked by the SoC designer. **While 268 of our generated security properties are covered in DV, 58 properties are not covered in the test cases, which clearly demonstrate that NSPG is adept at accounting for specifications that are not covered in DV.** This shows that our proposed framework, NSPG, is able to efficiently identify security properties in the documents. Next, we use these newly generated security properties to verify the bugs.

### C. Effectiveness in Discovering Violations

The extracted security properties provide us with essential information to construct various constraints when generating test cases for vulnerable IP designs. We choose to transfer the properties into SystemVerilog assertion format. Figure 7 shows an example of the process, transforming an extracted property into an assertion that can be applied for verification. It breaks the conjunctions of the sentence to identify the fragments of the operation relation. The nouns are transformed into RTL-level registers based on the IP register listing, and the verbs are converted into operators. Finally, the fragments rejoin to present a constraint for operation behavior, which can be asserted into the original RTL. Table VII outlines a few security properties generated by our framework and their corresponding CWE categories, from which we discover some vulnerabilities in the design. By creating constraints based on the acquired security properties and generating test cases on the design, we are able to detect eight vulnerabilities in the buggy IP designs. Table VIII demonstrates the details of eight bugs we identified from the extracted security properties. It presents the security objectives violated, CVSS score [6] (ranges from 0 to 10, a higher score refers to more severe vulnerabilities), CWE categories, and the potential security impacts on the system. We will discuss these vulnerabilities and their impacts on the system as follows.

**A. Key Manager:** The key manager implements the root key operation for the system and allows it to protect critical assets from malicious software. Two vulnerabilities are found in this IP: (1) The security property requires the key manager to wipe internal storage when it is in an invalid state. However, the implementation reverses the operation and wipes the key under a valid state. (2) It is required to continuously wipe the secret key with entropy during the operation state. How-

TABLE VIII: Eight bugs found in Key Manager, LC controller, HMAC, KMAC, and OTP memory controller, the violated security objectives, CWE, CVSS, and security impacts.

Vulnerability No.	Module	Sec Obj Violated	CWE Category	CVSS [6]	Security Impact
Bug 1 - Secret key is not wiped under a invalid state.	Key Manager	Device Integrity	Sensitive Information in Resource Not Removed Before Reuse - (226)	3.3	Information Leakage
Bug 2 - Secret key is not wiped during the operation state.	Key Manager	Device Integrity	Sensitive Information Not Removed Before Reuse - (226)	4.4	Information Leakage
Bug 3 - JTAG does not support bus integrity checks.	LC Controller	Device Integrity	Improper Protection for Outbound Error Messages and Alert Signals - (1320)	4	Unexpected Behavior
Bug 4 - The message byte order conversion is not operational.	HMAC	Device Integrity	Expected Behavior Violation - (440)	3.1	Unexpected Behavior
Bug 5 - Digest is not cleared when SHA is disabled.	HMAC	Device Integrity	Sensitive Information in Resource Not Removed Before Reuse - (226)	4.8	Information Leakage
Bug 6 - Key is not written with randomly generated value.	HMAC	Exploits from Software	Sensitive Information in Resource Not Removed Before Reuse - (226)	2.8	Unprivileged Access
Bug 7 - Software does not provide the key in masked form.	KMAC	Exploits from Software	Security Primitives and Cryptography Issues - (1205)	4.2	Information Leakage
Bug 8 - Lock control signal is bypassed with fault instruction.	OTP Controller	Unprivileged Code	Improper Access Control for Register Interface - (1262)	5.8	Unprivileged Access

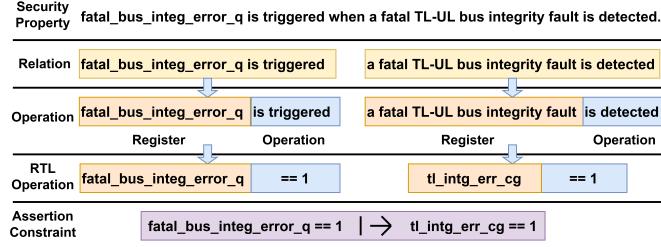


Fig. 7: Process of transforming a security property into an assertion. The first row breaks the sentences into relation conjunctions. The second row identifies the design components and the operations. The third row converts the texts into registers and operators. The forth row reconstructs them into the constraint for verification.

ever, the implementation does not replace the key registers, leaving sensitive information vulnerable. These vulnerabilities could potentially impact the confidentiality of the secret key, allowing the attacker to reveal the key information.

**B. LC Controller:** The life cycle (LC) controller controls the peripheral interactions on the chip interconnect bus. The security property requires the signal *fatal\_bus\_integ\_error\_q* to be set to one, when any bus integrity fault is detected. However, the boundary-scan test controller (JTAG) Test Access Port (TAP) does not provide a bus integrity check signal, which might cause integrity check failure in the life cycle controller and unexpected behavior in the system.

**C. HMAC:** The HMAC module implements a SHA-256 hash-based authentication module to ensure the integrity of any incoming message and its encryption code from the secret key. Three security bugs are found in this IP: (1) The first bug occurs when the software wants to convert the message byte order. It is required to set *CFG.endian\_swap* register to one. However, the implementation reverses the operation, making the IP convert the message, when *CFG.endian\_swap* is zero. This will cause unexpected operations due to incorrect instructions for converting messages. (2) The second bug occurs when the SHA engine is disabled. Although it is required to clear the digest in HMAC, no implementation is built to satisfy this property. (3) The third bug involves key randomization when the CPU writes values to the secret key. The key is required to be wiped with randomly generated value; however, no implementation is addressed for this property.

**D. KMAC:** The KMAC module is a Keccak-based message

authentication code generator used to verify the incoming messages. It utilizes masked permutations to prevent side-channel attacks. The security property requires the software to provide the key in masked form when the *EnMasking* parameter is not set and the *SwKeyMasked* parameter is set. However, this mechanism is not correctly implemented, leaving the software with an unmasked key. This could cause key leakage through an unprivileged software adversary.

**E. OTP Memory Controller:** The OTP memory controller is a module that provides a device with a one-time programming functionality. The security property only allows the IP to respond and write into the readout register when the lock control is not active. However, the IP is implemented with a mechanism that allows it to bypass the read-and-write lock control signal every four clock cycles. This will cause unpredictable behavior of the controller and allow the software adversary to attack the integrity of the module.

In summary, we have discovered eight vulnerabilities in five hardware IP designs from the 326 security properties we generated using NSPG. These vulnerabilities may cause information leakage, unexpected behavior, and unprivileged accesses in these IPs. It proves that these extracted security properties can provide valuable information to generate constraints for the hardware verification process.

## VI. DISCUSSION

**A. False Positives and False Negatives:** We will show some examples of the false positives and false negatives that were encountered, represented as FP and FN, respectively. A few of the FPs are short sentences that furnish equations or incomplete register interactions such as “How often FSM wakes up from ADC PD mode to take a sample, measured in always on clock cycles.” Although the framework is able to identify the operations, there is not enough information in the sentence to construct a complete security property. In contrast, FNs usually involve the structure of the sentences such as “To this end, the processor has to set the SIDELOAD bit in CTRL\_SHADOWED to 1”. Despite including two registers, the operation does not describe their behaviors separately, making it difficult for NSPG to classify them. Overall, the FP rate is 4%, and FN rate is 15% for the five IP documents. It should be mentioned that NSPG is the first work that utilizes NLP for hardware security property generation, and there are potentials for further improving its performance.

**B. Compared to text classification models:** We also explore standard text classification models such as TF-IDF and Bag-of-Words trained with  $\mathcal{D}_{cls}$  and tested on  $\mathcal{D}_{val}$ . Among them, TF-IDF achieved 70% accuracy for OpenTitan, and 61% accuracy for RISC-V, Bag-of-Words achieved 51% accuracy for OpenTitan, and 53.12% for RISC-V. Hence, we can conclude that HS-BERT is significantly more effective than standard text classifiers.

**C. Compared to ChatGPT:** In this section, we compare NSPG against ChatGPT [4], a popular chatbot based on the OpenAI Generative Pre-trained Transformer (GPT)-3.5 language model that is adept at generating human-like text responses to any input in a conversational context [44]. We investigated if such a state-of-the-art off-the-shelf general LLM can outperform a small domain-specific BERT model in solving the security property identification task. We evaluated their performances on a reduced dataset of 50 sentences, which contained 25 property-related and 25 non-property-related sentences (the dataset size was necessitated by resource limitations, since GPT-3.5 is not publicly available as an open-source model). ChatGPT’s evaluation resulted in numerous false positives and false negatives, with only 35 sentences correctly classified, many of which were unsuitable for SoC security verification. For example, sentences like “The ADC is continually powered on” and “In addition, they could potentially also be extracted when being transferred over the TL-UL bus interface” were incorrectly classified as properties. Conversely, sentences such as “For encryption or if the mode is set to CFB, OFB, or CTR, there is no such initial delay upon changing the key” and “The AES unit cannot recover from such an error and needs to be reset” were labeled as non-properties. The accuracy, recall and F1-score obtained by the ChatGPT model were 68%, 88% and 73%, respectively. On the contrary, NSPG outperforms ChatGPT by identifying all 25 property-related sentences, thereby furnishing an accuracy, recall, and F-1 score of 100%, respectively.

## VII. RELATED WORK

This section presents prior related work proposed for hardware security property generation and verification.

### A. Hardware Security Verification Approaches

*Information Flow Tracking* (IFT) improves hardware security by identifying malicious input interfaces, regulating the use of counterfeit information, and verifying hardware operation flows [13], [14], [42], [43], [47], [48]. This approach has been demonstrated to effectively detect malicious attacks with security property-generated constraints. *Theorem proving* utilizes a language, VeriCoq, to automate the transformation of RTL into Coq theorem. This approach validates the converted language through the Proof-Carrying Hardware Intellectual Property (PCHIP) framework and ensures the robustness of a third-party IP with formal proofs of security properties [18], [36]. *Assertion-based security verification* employs formal and simulation-based methods to detect violations against assertions included in RTL designs [19], [30]. *Property-specific*

*information flow* utilizes property specifications to generate information flow models that are catered to the pre-defined security properties. The models are verified by conducting a property-specific search and identifying security-critical paths [31]. *Directed test generation* has been developed to address state space explosion. Existing research, such as [11], [37], utilize control flow graphs on RTL models to ensure functional correctness. Symbolic techniques incorporating security properties and concolic testing can detect violations in RTL and conclusively verify designs [39], [50].

### B. Property Generation Techniques

A recent approach, *Isadora*, creates information flow specifications of core designs and combines IFT and security specifications [22]. It requires conclusive testbenches and simulation traces for property generation. Therefore, it might be not scalable when applied to more complex SoCs. PCHIP introduces the concept of theorem generation functions, which enables the generation of security theorems independent from the information flow traces, thereby assisting in the development of data secrecy properties [17], [33]. However, PCHIP is only limited to cryptographic circuits. *SCIFinder* is a recent approach that generates Security-Critical Invariants (SCI) for design verification [51]. This approach only studies a limited size of security properties and is not scalable when applied to property generation on more complex processors. *Comments-based property generation* is an NLP-based translation technique was developed, which automatically generates Computation Tree Logic (CTL) verification properties from Hardware Description Language (HDL) *code comments* [29]. In comparison, NSPG uses a BERT model to automatically identify and generate new security property-related sentences from the documentation. Furthermore, [29] has been evaluated only on small *well-commented* benchmarks.

## VIII. CONCLUSION

This paper presents NSPG, the first NLP-based automated hardware security property generation method, that utilizes SoC documentation to extract security properties. NSPG includes a novel hardware security-specific language model (HS-BERT) and a data modification technique to improve automated security property generation. Our framework is evaluated on OpenTitan SoC documentation, resulting in 326 correctly extracted security properties from 1723 sentences for five hardware IPs. Furthermore, these security properties help discover eight vulnerabilities in a buggy design, which proves the effectiveness of the generated security properties. Additionally, our evaluations prove that NSPG furnishes better performance than ChatGPT, a popular chatbot system, for SoC security property generation. With the advent of LLMs, we envision that NSPG will lay the foundation for utilizing NLP approaches in SoC design and verification.

## IX. ACKNOWLEDGEMENT

This research is partially supported by Technology Innovation Institute, Abu Dhabi, United Arab Emirates.

## REFERENCES

- [1] “Cwe - cwe-1194: Hardware design (4.0),” <https://cwe.mitre.org/data/definitions/1194.html>, (Accessed on 05/15/2020).
- [2] “Github - shal/cppdoc: Powerful documentation generation tool for c++,” <https://github.com/shal/cppdoc>, (Accessed on 01/12/2023).
- [3] “Hack@dac22 – hack@event hw ctf,” <https://hackatevent.org/hackdac22/>, (Accessed on 01/26/2023).
- [4] “Introducing chatgpt,” <https://openai.com/blog/chatgpt>, (Accessed on 04/12/2023).
- [5] “Javadoc tool home page,” <https://www.oracle.com/java/technologies/javase/javadoc-tool.html>, (Accessed on 01/12/2023).
- [6] “NVD - CVSS v3 calculator,” <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>, (Accessed on 02/05/2023).
- [7] “Opentitan — opentitan documentation,” <https://docs.opentitan.org/>, (Accessed on 01/26/2023).
- [8] “Questa Secure Check - Exhaustive Verification of Secure Paths — Siemens Software,” <https://eda.sw.siemens.com/en-US/ic/questa/formal-verification/secure-check/>, (Accessed on 02/05/2023).
- [9] “Risc-v instruction set specifications — riscv-isa-pages documentation,” <https://msyksphinz-self.github.io/riscv-isadoc/html/index.html>, (Accessed on 01/27/2023).
- [10] “Tortuga Logic — Synopsys,” <https://www.synopsys.com/dw/ipdir.php?ds=arc-access-member-tortuga-logic>, (Accessed on 02/05/2023).
- [11] A. Ahmed, F. Farahmandi, and P. Mishra, “Directed test generation using concolic testing on rtl models,” in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1538–1543.
- [12] W. Ammar, D. Groeneweld, C. Bhagavatula, I. Beltagy, M. Crawford, D. Downey, J. Dunkelberger, A. Elgohary, S. Feldman, V. Ha *et al.*, “Construction of the literature graph in semantic scholar,” *arXiv preprint arXiv:1805.02262*, 2018.
- [13] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, “Register transfer level information flow tracking for provably secure hardware design,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1691–1696.
- [14] A. Ardeshiricham, Y. Takashima, S. Gao, and R. Kastner, “Verisketch: Synthesizing secure hardware designs with timing-sensitive information flow properties,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1623–1638.
- [15] I. Beltagy, K. Lo, and A. Cohan, “Scibert: A pretrained language model for scientific text,” *arXiv preprint arXiv:1903.10676*, 2019.
- [16] B. Bentley, “Validating the intel pentium 4 microprocessor,” in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 244–248.
- [17] M.-M. Bidmeshki, X. Guo, R. G. Dutta, Y. Jin, and Y. Makris, “Data secrecy protection through information flow tracking in proof-carrying hardware ip—part ii: Framework automation,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2430–2443, 2017.
- [18] M.-M. Bidmeshki and Y. Makris, “Vericoq: A verilog-to-coq converter for proof-carrying hardware automation,” in *ISCAS*. IEEE, 2015, pp. 29–32.
- [19] M. Bilzor, T. Huffmire, C. Irvine, and T. Levin, “Evaluating security requirements in a general-purpose processor by combining assertion checkers with code coverage,” in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 2012, pp. 49–54.
- [20] Cadence, “Jaspergold formal verification platform (apps),” [https://www.cadence.com/en\\_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html](https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html), (Accessed on 05/15/2020).
- [21] G. Dessouky, D. Gens, P. Haney, G. Persyn, A. Kanuparthi, H. Khattri, J. M. Fung, A.-R. Sadeghi, and J. Rajendran, “{HardFails}: Insights into {Software-Exploitable} hardware bugs,” in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 213–230.
- [22] C. Deutschbein, A. Meza, F. Restuccia, R. Kastner, and C. Sturton, “Isadora: Automated information flow property generation for hardware designs,” in *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, 2021, pp. 5–15.
- [23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [24] N. Farzana, F. Rahman, M. Tehrani, and F. Farahmandi, “Soc security verification using property checking,” in *2019 IEEE International Test Conference (ITC)*. IEEE, 2019, pp. 1–10.
- [25] E. A. Felix and S. P. Lee, “Systematic literature review of preprocessing techniques for imbalanced data,” *IET Software*, vol. 13, no. 6, pp. 479–496, 2019.
- [26] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy, “A survey of data augmentation approaches for nlp,” *arXiv preprint arXiv:2105.03075*, 2021.
- [27] A. Gillioz, J. Casas, E. Mugellini, and O. Abou Khaled, “Overview of the transformer-based models for nlp tasks,” in *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2020, pp. 179–183.
- [28] S. Gupta and A. Gupta, “A set of measures designed to identify overlapped instances in software defect prediction,” *Computing*, vol. 99, no. 9, pp. 889–914, 2017.
- [29] C. B. Harris and I. G. Harris, “Generating formal hardware verification properties from natural language documentation,” in *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*. IEEE, 2015, pp. 49–56.
- [30] M. Hicks, C. Sturton, S. T. King, and J. M. Smith, “Specs: A lightweight runtime mechanism for protecting software from security-critical processor bugs,” in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015, pp. 517–529.
- [31] W. Hu, A. Ardeshiricham, M. S. Gobulukoglu, X. Wang, and R. Kastner, “Property specific information flow analysis for hardware security verification,” in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.
- [32] T. Ilion, C.-N. Anagnostopoulos, M. Nerantzaki, and G. Anastassopoulos, “A novel machine learning data preprocessing method for enhancing classification algorithms performance,” in *Proceedings of the 16th International Conference on Engineering Applications of Neural Networks (INNS)*, 2015, pp. 1–5.
- [33] Y. Jin, X. Guo, R. G. Dutta, M.-M. Bidmeshki, and Y. Makris, “Data secrecy protection through information flow tracking in proof-carrying hardware ip—part i: Framework fundamentals,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2416–2429, 2017.
- [34] B. Krawczyk, M. Woźniak, and G. Schaefer, “Cost-sensitive decision tree ensembles for effective imbalanced classification,” *Applied Soft Computing*, vol. 14, pp. 554–562, 2014.
- [35] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, “Biobert: a pre-trained biomedical language representation model for biomedical text mining,” *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.
- [36] E. Love, Y. Jin, and Y. Makris, “Proof-carrying hardware intellectual property: A pathway to trusted module acquisition,” *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 25–40, 2011.
- [37] Y. Lyu and P. Mishra, “Automated test generation for activation of assertions in rtl models,” in *ASP-DAC*, 2020.
- [38] A. Maña and G. Pujol, “Towards formal specification of abstract security properties,” in *2008 Third International Conference on Availability, Reliability and Security*. IEEE, 2008, pp. 80–87.
- [39] X. Meng, S. Kundu, A. K. Kanuparthi, and K. Basu, “Rtl-contest: Concolic testing on rtl for detecting security vulnerabilities,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 466–477, 2021.
- [40] G. A. Miller, “Wordnet: a lexical database for english,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.
- [41] R. C. Moore and W. Lewis, “Intelligent selection of language model training data,” in *Proceedings of the ACL 2010 conference short papers*, 2010, pp. 220–224.
- [42] J. Oberg, W. Hu, A. Irturk, M. Tiwari, T. Sherwood, and R. Kastner, “Theoretical analysis of gate level information flow tracking,” in *Proceedings of the 47th Design Automation Conference*, 2010, pp. 244–247.
- [43] M. Qin, X. Wang, B. Mao, D. Mu, and W. Hu, “A formal model for proving hardware timing properties and identifying timing channels,” *Integration*, vol. 72, pp. 123–133, 2020.
- [44] J. Schulman, B. Zoph, C. Kim, J. Hilton, J. Menick, J. Weng, J. F. C. Uribe, L. Fedus, L. Metz, M. Pokorny *et al.*, “Chatgpt: Optimizing language models for dialogue,” *OpenAI blog*, 2022.
- [45] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. [Online]. Available: <https://aclanthology.org/P16-1162>

- [46] J. Stefanowski, “Dealing with data difficulty factors while learning from imbalanced data,” in *Challenges in computational statistics and data mining*. Springer, 2016, pp. 333–363.
- [47] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, “Secure program execution via dynamic information flow tracking,” *ACM Sigplan Notices*, vol. 39, no. 11, pp. 85–96, 2004.
- [48] M. Tiwari, H. M. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, “Complete information flow tracking from the gates up,” in *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, 2009, pp. 109–120.
- [49] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *arXiv preprint arXiv:1609.08144*, 2016.
- [50] R. Zhang, C. Deutschbein, P. Huang, and C. Sturton, “End-to-end automated exploit generation for validating the security of processor designs,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 815–827.
- [51] R. Zhang, N. Stanley, C. Griggs, A. Chi, and C. Sturton, “Identifying security critical properties for the dynamic verification of a processor,” *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 541–554, 2017.