CPSC 490 Senior Project:

Using Predicate Logic and LLMs to Assess Compliance with the Computer Fraud and Abuse Act

Rosie Rothschild

Faculty advisors: Professor Ruzica Piskac

May 2, 2024

Abstract

This project builds upon existing methods to translate the Computer Fraud and Abuse Act (CFAA) to predicate logic and creates a user-friendly GUI to interact with the translation. The law can be difficult to understand, and current AI chatbot solutions aren't always accurate; this solution uses formal verification to ensure that the law is interpreted as precisely as possible. The translation was completed using a combination of manual effort and LLMs. Then, the user could assert the details of a particular situation to be appended to the translation, where I ran the Z3 algorithm as an SMT solver to determine whether the given situation infringed on the CFAA. If a situation broke the CFAA, I utilized the unsat-core function to determine which assertions infringed upon the law so a user could understand why their conduct was illegal. The Z3 processes ran locally using the command line and lived on a local Node.js Express server, which could be reached using a POST request. I built the sleek front-end design using the React.js framework. Much of the complexity in the project stemmed from determining the specific questions to ask the user to obtain the necessary information to run the appended situation without making assumptions or becoming overly repetitive. After obtaining input from the user, it calls the Node.js backend using a POST request, which returns whether the situation breaks the CFAA and, if so, in what way. It's important to acknowledge that the translation doesn't account for references outside the law nor subjective statements open to interpretation. This project has broad implications, as it builds on the promise of using formal verification to understand complex bodies of law and make them more accessible.

Contents

1 Introduction **3**

2 Background **3**

3 Design and Process **5**

3.1 Part 1: Translation 5

3.2 Part 2: SMT Solver, Backend Implementation, and Unsat Core 7

3.3 Part 3: GUI 8

4 Illustrative Example **10**

5 Brief LLM Hardware Power Exploration **12**

6 Discussion and Future Work **13**

7 Acknowledgements **16**

1 Introduction

While computer science is governed by the laws of code, the real world is governed by our legal code. This thesis builds upon methods of translation from legal code to predicate logic to determine whether situations comply with the US legal code. Specifically, it explores the Computer Fraud and Abuse Act (CFAA), which is a section of the US legal code that makes it illegal to "intentionally access a computer without authorization or exceeds authorized access" (Section 18 U.S. Code § 1030). [CFA] With hacking and cyber security breaches becoming more common, having an enforceable legal code to prosecute this abuse is essential.

This project aims to understand better how LLMs and other tools can formally verify the results they output. Utilizing a translation from law to logic is a promising way to solve this problem.

2 Background

Congress enacted the Computer Fraud and Abuse Act in 1986 to federally prosecute hacking crimes to obtain government information. The law has been amended in 1989, 1994, 1996, 2001, 2002, and 2008 to keep up with modern technology and broaden the scope of illegal conduct. Additionally, the Supreme Court recently specified the meaning of "exceeds authorized access" in *Van Buren v. United States* after a decades-long split among the circuits. With this new clarification of the law, the CFAA became more narrowly defined, applying only to instances where a person enters "particular areas of the computer– such as files, folders, or databases– that are off limits to him." (*Van Buren v. US*) [van21]. The court goes on to explain that liability under the CFAA "stems from a gates-up-or-down inquiry– one either can or cannot access a computer system, and one either can or cannot access certain areas within the system." (*Van Buren v. US*). So, one can only violate the CFAA if someone trespasses through a closed gate. (Kirr) [OK21] [PLB16]. Practically, it can be difficult to determine what falls under the CFAA, as the Supreme Court does not specify what a closed gate looks like. By reimagining the CFAA, a law utilized to regulate code, as a predicate logic code, we can better understand what situations comply with this statute and what

situations would violate it.

A body of work has been done to translate legislation into predicate logic to apply to real-life situations (Verhij)[BV03]. The law comprises argumentation schemes that are context-dependent and contingently valid, making it difficult to translate to an objective code. Verheij presents a method to model these schemes formally. Some work has been done building upon this model to formalize private international law and contracts (Minh Dung) [PMD09] [PMD10]. There has been no work on applying legal formalized logic to the CFAA or more complex cross-referencing laws. My thesis will explore mechanisms to translate this complex legal code, which will help to build the foundation of translation for even more complicated sections of the law in the future.

This project also utilizes Satisfiability Modulo Theories (SMT) solvers to determine whether a given situation complies with the CFAA. An SMT solver is a tool that can be used to solve decision problems for logical formulas, which may involve various combinations of theories such as arithmetic, arrays, bit-vectors, and quantifiers (De Moura)[DMB08]. An SMT solver extends the abilities of a SAT solver to include capabilities to handle data types and operations beyond Boolean logic. Given a logical formula, such as the law to logic translation, an SMT solver can determine whether it's satisfiable. By asserting that the formula doesn't violate the CFAA in the translation, we can see whether the user's assignment of variables allows the formula to hold still true. SMT solvers allow us to check for compliance automatically. Furthermore, if a formula cannot be satisfied (i.e., is unset), the SMT solvers can state which assertions create a conflict. This functionality enables the user to understand why their actions or situations violate the CFAA.

This project uses the Z3 algorithm as an efficient SMT solver, as it is a publicly available tool created by Microsoft Research and works extremely efficiently (De Moura)[DMB08]. The Z3 command line tool can also directly run SMT-LIB files, which provide a standardized representation of logical problems. All translations in this project are represented in this way.

3 Design and Process

This project will solve the following two problems: translating complex legal code into predicate logic and determining the application of the CFAA. The CFAA can be difficult to apply in various instances, and the penalties can change based on subtle distinctions in a given situation. This project formalizes the law in predicate logic to determine how to objectively apply the statute depending on the facts of a case. There has yet to be a translation from CFAA to first-order logic or significant work on legal code with a high level of complexity. Since the CFAA includes cross-referencing within the section, a non-trivial implementation of First Order Logic will be required. This will be a step toward translating more complicated sections of law.

In this project, I translated parts (1)-(6) of subsection (a) of the CFAA to predicate logic in the SMT-LIB format. Then, I created a GUI using React for a user to input details about a given situation that may or may not violate parts of the CFAA. I used Node.js to create a back-end server that appends the given details of the situation to the translation and utilizes Z3 to determine whether the situation violates the CFAA. If so, I utilize Z3 to determine which assertions are responsible for the violation so the user understands why their conduct is illegal. This section will dive into each step, where I will outline the various challenges and lessons I learned while building this tool.

3.1 Part 1: Translation

The first step in the design of this project is the translation process from law to logic. The translation can be performed manually by analyzing the law and synthesizing it into first-order logic. This was the primary mechanism I used for this project since the law is mostly self-contained and finite. I read each section of the law line by line and meticulously broke down each statement into smaller parts; each part was represented by a unique Boolean constant. I quickly realized that because my project was building a translation for one instance of a scenario, I could utilize boolean logic to simplify the process. These parts were then assembled into logical implications, according to the CFAA. The manual translation process was slow and tedious, yet it allowed me to deeply understand the intricacies of the law and broaden my understanding of boolean logic.

We can translate the following statement into FOL as shown below.

> *Whoever –*
> *(1) having knowingly accessed a computer without authorization or exceeding authorized access, and by means of such conduct having obtained information that has been determined by the United States Government pursuant to an Executive order or statute to require protection against unauthorized disclosure for reasons of national defense or foreign relations, or any restricted data, as defined in paragraph y. of section 11 of the Atomic Energy Act of 1954, with reason to believe that such information so obtained could be used to the injury of the United States, or to the advantage of any foreign nation willfully communicates, delivers, transmits, or causes to be communicated, delivered, or transmitted, or attempts to communicate, deliver, transmit or cause to be communicated, delivered, or transmitted the same to any person not entitled to receive it, or willfully retains the same and fails to deliver it to the officer or employee of the United States entitled to receive it;*
> *...*
> *shall be punished as provided in subsection (c) of this section.*

Assuming all Boolean constants have previously been declared, this would match with the following assertions:

> 1. (KnowinglyAccessesComputerWithoutAuthorization $\vee$ KnowinglyAccessesComputerExceedingAuthorization) $\wedge$ ObtainsRestrictedInformation $\implies$ NotCompliant
> 2. InformationDeterminedToRequireProtectionAgainstDisclosures $\wedge$ (InformationCanBeUsedToInjureUS $\vee$ InformationCanBeUsedToAdvantageForeignNation $\vee$ CommunicatesInformationToUnauthorizedPerson $\vee$ FailedToDeliverToUSEntity) $\implies$ ObtainsRestrictedInformation

We can trace each Boolean in the assertions to specific fragments of the section of the law. The first 5 parts (1)-(6) of subsection (a) of the CFAA were translated similarly. However, it is important to note that certain aspects of this law were out of the scope of my project. For instance, I do not refer to section 11 of the Atomic Energy Act to

further define what kind of information would require protection according to the US government. The translation, instead, assumes this will have already been considered when the user inputs the details of their situation. The translation also doesn't cover part (a)(7) of the law, as this part tackles extortion, which was less relevant to the inputted scenarios.

In many other instances, it will not be feasible to perform a manual translation due to the length of the text or the complex cross-referencing across different resources. Generating a generic method for translation from law, or English text, to logic is currently a highly researched question. For instance, if someone wanted to translate the entire US legal code, this could not be done by hand, as it is far too long and contains vast references throughout the body of information. Instead, more complex strategies, such as using Large Language Models (LLMs), will be necessary. Since the LLMs are trained on extremely large data sets, they can recognize patterns in the text and convert them rapidly into logic. While this is very efficient, it can produce significant errors, and the results aren't always usable.

Because of the inaccuracies in directly translating to SMT-LIB translations using LLMs, Professor Piskac and her team, among other researchers, are trying to use an intermediate step in the process. A third translation mechanism explored translates from law to an intermediary representation before converting into pure logic. For instance, we can use a Python class representation of a scenario. This can be deeply beneficial, as LLMs are more familiar with Python than SMT languages. Ultimately, however, my project just required the translation of the CFAA, so to prioritize accuracy, I completed the translation by hand.

3.2 Part 2: SMT Solver, Backend Implementation, and Unsat Core

After obtaining the translation, the next step was to use an SMT solver to determine whether a given input broke the CFAA. To do this, I always defined the `NotCompliant` boolean as `false`, which means I asserted that the user had not broken the CFAA. If the Z3 algorithm could find a satisfiable model given the assertion that the given situation was compliant with the CFAA, then the user's inputted situation did not

violate the law. However, if there was an inherent contradiction between the assertion that the CFAA was not violated and one of the user's inputs, the user must have broken the law. The SMT solver automatically checks compliance for us by design. While the translation laid the foundation to run each simulation, I had to create a backend to receive the user's inputs, append them to the SMT-LIB file, run the file, and report back whether the situation was compliant. If the situation was not compliant, I then appended the `get-unsat-core` command to report to the user which specific assertions resulted in the user violating the CFAA.

I decided to build a local express Node.js server to handle the backend logic. I created a POST API to handle receiving the user's inputted values. Next, I went through each input, and for those that weren't `null`, I added an assertion to the bottom of the SMT-LIB translation to assert whether the user inputted the value as `true` or `false`. I copied the original translation to a new executable file to add the assertions and ensure the initial translation remained intact. Each assertion was given a name that could be referenced later in the program. Finally, I added `(assert (!(not NotCompliant) :named notComp))` as described above and `(check-sat)` before running the appended situation. Depending on whether the algorithm could find an assignment to satisfy the given modulo, the program would return `sat` or `unsat` to `stdout`, which the Node.js request would then use to return information to the caller. Since the translation was already in the SMT-LIB form, the Z3 algorithm, using the command line, worked well when running the simulation.

If the program returned `unsat`, I created an additional file and appended the `(get-unsat-core)` command as defined earlier. My program then spawned a new process to use the Z3 command to run the file, and the conflicting assertions were returned through `stdout`. After splitting and splicing the string, an array of faulty assertions was returned to the caller. The backend aspect of the project required me to create clever ways to write to files, spawn child processes, and organize information in clear, concise ways.

3.3 Part 3: GUI

The last step of this process was creating an interface for the user to input a scenario. I decided to build a React.js web app for the front end since the state man-

agement allowed me to easily keep track of the complex inputs and implement a sleek design. Building the front end involved multiple steps – deciding what questions the user would answer (and how they would do so), updating the state of the user inputs, calling the Node.js backend to determine compliance, and outputting the results.

The most complicated aspect of building the UI was determining the best questions to ask the user to efficiently determine the necessary inputs to each situation. Many of the inputs relied on answers to previous questions, so determining the order and flow of questions was extremely important. Not every question is initially shown, as some may not be relevant to the situation. For instance, if no information was obtained in the situation, there's no need to inquire what information was gathered from the user. This involved multiple challenges, as I needed to set 31 input values without requiring redundancy for the user. Ultimately, I could determine the inputs (`null`, `true`, or `false`) given the following set of questions:

Question	**Booleans**
1. Did you knowingly or intentionally access a computer you didn't have authorization for?	KnowinglyAccessesComputerWithoutAuthorization, IntentionallyAccessesComputerWithoutAuthorization
2. Did you knowingly access a computer that you had authorization for but exceeded the scope of the authorization?	KnowinglyAccessesComputerExceedingAuthorization, IntentionallyAccessesProtectedComputerWithoutAuthorization
3. What kind of computer did you access?	ComputerExclusivelyForGovernmentUse, NonpublicComputerOfUSDepartmentOrAgency
4. If the computer is not exclusively used by the government or financial institution, did the conduct of the situation affect the government/financial use?	ConductAffectsGovernmentUse, IsProtectedComputer, InformationFromProtectedComputer
5. In the computer you accessed, did your conduct cause damage:	ConductResultsInDamageAndLoss, ConductResultsInRecklessDamage
6. Did you obtain information as a result of accessing the computer?	N/A
7. What kind of information did you obtain?	InformationDeterminedToRequireProtection, InformationCanBeUsedToInjureUS, InformationCanBeUsedToAdvantageForeignNation, ContainsFinancialRecordofFinancialInstitution, ContainedInConsumerReportingAgency
8. Did you share the information with anyone unauthorized?	CommunicatesInformationToUnauthorizedPerson
9. Did your actions prevent information delivery to a US entity?	FailedToDeliverToUSEntity
10. Did you have intent to commit fraud when accessing the computer?	IntentToDefraud
11. Did you obtain anything of monetary value through the fraud?	ObtainsAnythingOfValue, ObjectOfFraudAndThingObtainedOnlyUseOfComputer
12. Was the value of your computer use worth more than $5000?	ValueOfUseOfComputerNotMoreThan5000
13. Did you knowingly cause the transmission of a program, information, or code, which caused damage to a protected computer?	KnowinglyCausesTransmissionOfCode, TransmissionCausesDamageWithoutAuthOfProtectedComputer

Table 1: Questions asked to the user and the booleans within the Translation that each question answers

After determining the questions I needed to ask the user, I designed a mock-up of what this may visually look like using Figma. I utilized React Bootstrap libraries to aid in the visual design and implemented a state variable for each necessary input. These inputs were then delivered to the Node.js backend as described in Part 2, and the situation was deemed compliant with the CFAA or illegal. When the situation was illegal, the front end would match the specific assertion number with a readable statement for the user to understand what part of their actions were illegal. Building the front end was an exciting opportunity to get comfortable with a large and complex code base, practice good state management, and work on my UI design skills.

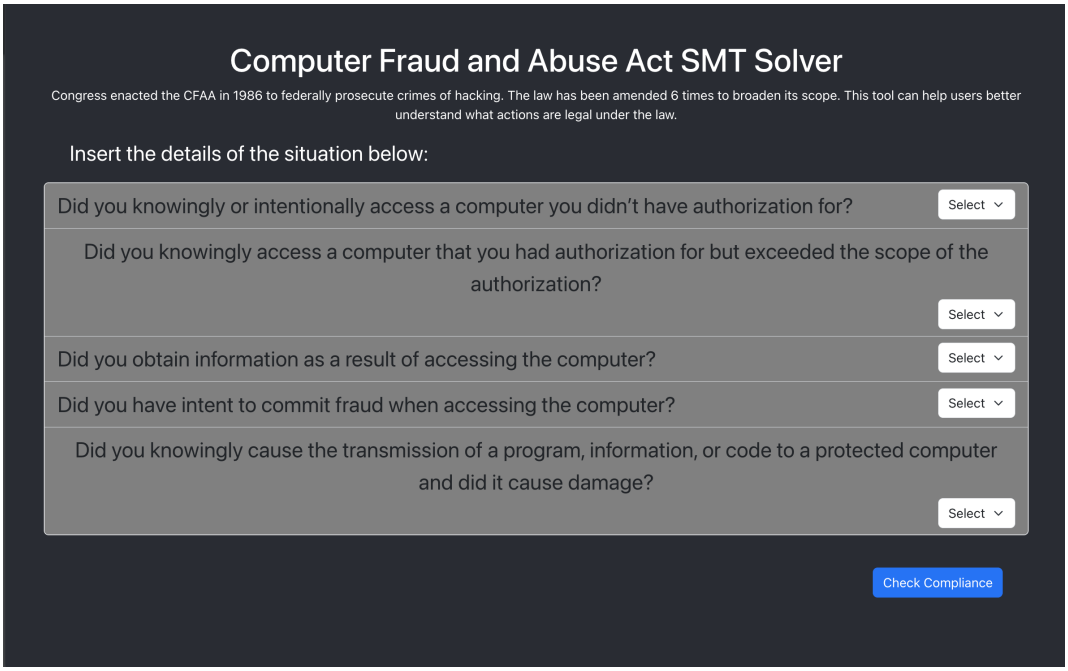

Figure 1: My GUI before checking for compliance

4 Illustrative Example

This section will walk through a brief motivating example of how an individual can utilize my app. Suppose Bob recently attempted to hack into a computer before realizing he may be guilty of a federal crime. Bob had the authorization to access some parts of the computer but exceeded his access and hacked into the special files of his bank's computer. He obtained financial information about the bank's customers that he should not have been privy to, but he didn't retain or share it with anyone else.

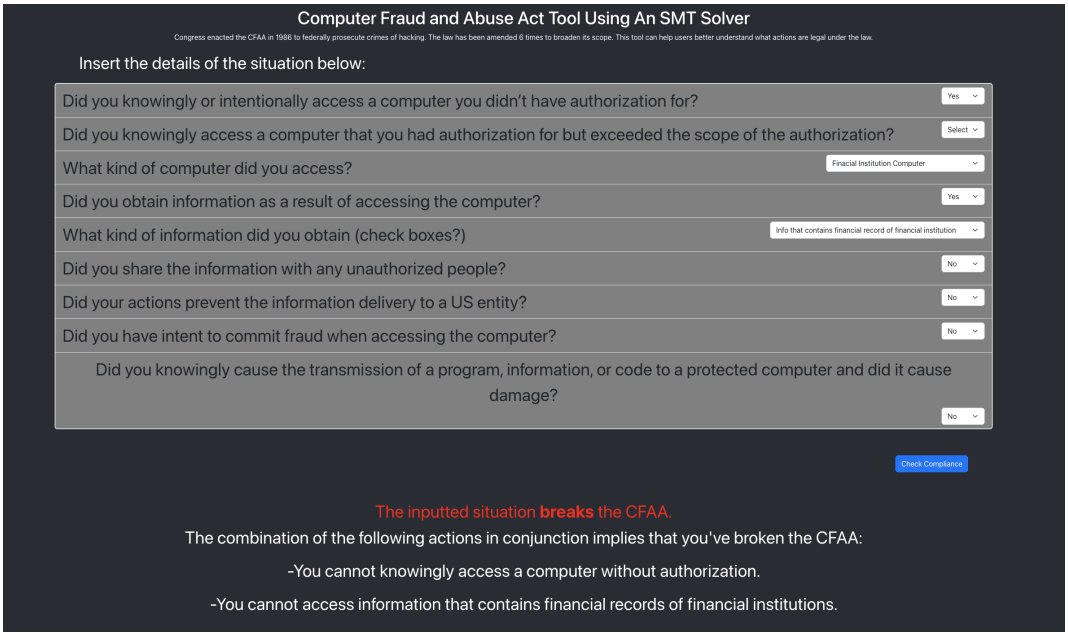

Figure 2: GUI output when situation is illegal. Note: the window is zoomed out to fit all content, this is not an accurate font size.

Bob's conduct also did not impact the function of the bank's computer in any way, nor did he transmit any code or commit fraud with what he gathered. Bob wants to know if his actions violate the CFAA.

Bob used the web app to determine whether he should be worried about law enforcement coming after him. Bob responds "Yes" to knowingly exceeding access, utilizing a computer from a financial institution, and obtaining information that includes financial records.

In this situation, even though Bob didn't cause any damage, his conduct still violates the CFAA. According to subsection a2, "Whoever– intentionally accesses a computer without authorization or exceeds authorized access, and thereby obtains— information contained in a financial record of a financial institution ... shall be punished." This translates in my logical code as the following:

1. (KnowinglyAccessesComputerWithoutAuthorization $\vee$ KnowinglyAccessesComputerExceedingAuthorization) $\wedge$ ObtainsRestrictedInformationSection2 $\implies$ NotCompliant
2. (ContainsFinancialRecordofFinancialInstitution $\vee$ ContainedInConsumerReportingAgency) $\implies$ FinanciallyRestrictedInformation
3. (FinanciallyRestrictedInformation $\vee$ InformationFromDepartmentOrA-

gencyOfUS ∨ InformationFromProtectedComputer) ⟹ ObtainsRestrictedInformationSection2

KnowinglyAccessesComputerExceedingAuthorization and (Information) ContainsFinancialRecordofFinancialInstitution will both be asserted as `true`, which will in turn imply that the information gathered is FinanciallyRestrictedInformation and Bob ObtainsRestrictedInformationSection2. Therefore, these two factors imply that NotCompliant must be `true` as well; however, we've asserted that NotCompliant is `false` if the situation does not violate the CFAA. The SMT solver shows that this description cannot be satisfied, and Bob must have broken the law.

My app would respond with the following statements:

> The inputted situation breaks the CFAA.
> The combination of the following actions in conjunction implies that you've broken the CFAA:
> -You cannot knowingly exceed authorized access to a computer.
> -You cannot access information containing financial institutions' financial records.

5 Brief LLM Hardware Power Exploration

Because this project utilizes LLMs for part of the translation, and the eventual goal is to create formally verifiable LLMs, I will briefly analyze LLM power utilization and hardware. Machine learning, generative AI, and LLMs are experiencing a large increase in popularity worldwide. With the rise of LLM chatbots for everyday consumers and enterprise businesses, this demand will only increase in the coming years. The demand for AI also places a strain on power, as it takes tremendous energy in both the training and inference phases. While more research has been done on the energy it takes to train a model, Google reported that 60% of AI-related energy use came from the inference phase (de Vries) [dV23]. By estimating NVIDIA server sales, estimates project that they could use 85 to 134 terawatt hours (TWh) each year, which is around 0.5% of the world's current electricity demand. The same study found that "a single LLM interaction may consume as much power as leaving a low-brightness LED light bulb on

for one hour" (de Vries) [dV23]. Generative AI will only grow in the coming years, so ongoing research exists to increase efficiency and reduce energy consumption (Verdecchi) [VSC23].

The amount of energy used during training is directly related to the size and complexity of the input data. LLMs that require billions or trillions of parameters will use significantly more energy than smaller models that don't rely on such intensive computing resources. Hardware manufacturers aim to specialize their GPUs further and create custom-built hardware accelerators (ex, Google's Tensor Processing Units). Other unique algorithms and techniques continue to be explored, such as optical computing, computation with oscillatory neural networks, and trying new hardware resources such as processing-in-memory (PIM) architecture systems for graphical neural network training (Ogbogu) [OAD+23]. It's important to note that as energy efficiency improves, the resource demand may increase, ultimately having little to no effect on the total consumption of AI models (de Vries) [dV23]. However, decreasing LLMs power consumption will be essential to avoiding overwhelming the world's power sources and reducing emissions.

Outside of the hardware power analysis, it's interesting that this project has a clear connection to Boolean circuits. Because the translation was created using plain Boolean logic, we could also create a hardware simulation to determine whether a given situation was SAT. This is simply a further application of the Circuit SAT (CSAT) problem, an exciting connection to my work. In the future, I hope to implement the translation circuit using Verilog or an FPGA board.

6 Discussion and Future Work

This project successfully translated from the CFAA to predicate logic and provided a user-friendly GUI to interact with the translation. This builds upon previous work translating law and other text into logic, ensuring formal output verification. It is important to note certain limitations that arose during the project. Some parts of the law were not self-contained and became out of scope for the project. Additionally, many aspects of the law are open to interpretation and require more complicated analysis to determine whether or not someone violates the CFAA. For instance, what does it

mean to "knowingly exceed authorization" when entering a computer? It's difficult to understand if someone knew what they were doing in a particular situation and likely would require a jury to determine the answer to this question in a serious matter. The app also asks users about their "intent" when accessing systems, which again is extremely difficult to prove and can have various standards depending on the situation. Many of the questions the app asks are subjective, so it's important for users to fully understand that the program's output may not align with the legal system. The result is only as good as the inputted data. In the future, I'd like to break down each question further to limit the burden placed on the user. Having an app that clearly defines the dozens of gray areas left for interpretation would be far more useful. I also want to note that the project does not address the portion of the CFAA that handles threats, nor does it determine the punishments for any violation. Given the allotted time and the limitation that I could only ask specific questions with predetermined answers to users, I believe my solution works well. However, I hope to explore the remaining open questions.

This project aimed to create an app that reported reliable, formally verified information in the hopes that this type of verification could be applied to complex LLMs and chatbots. Formal verification is extremely important when discussing sensitive and important topics such as legal codes, and this project helps us get a step closer. More work must be done to develop a reliable way to automatically generate a translation from text to logic; however, once this ability develops, we can create an accessible chatbot to help individuals understand complex legal code. It is important to note, however, the reality that there will always be limitations to formally verifiable legal chatbots. The output is only as effective as the translation itself and as accurate as the user's inputs are. If the chatbot prompts a user to input the information to a subjective question, the answer hinges on the user's interpretation, which may vary from the legal decision. Because of this, more objective statutes and policies may be better suited for this sort of chatbot.

Overall, this project was a success and will help people better understand the intricacies of the CFAA. This project has substantially contributed to the progress of law to logic translation by building a larger dataset of legal code and SMT-LIB translation files for LLMs to train further. This project challenged me to broaden my understanding of

law and logic and improve my full-stack engineering skills to build a robust application.

7 Acknowledgements

I've always been fascinated by the intersection between computer science and law, and I'm grateful to have found the space in my senior project to explore it further. Thank you to my fearless advisor, Professor Ruzica Piskac, for all her help and support along my journey to complete this project. This project also would not have been possible without the support of PhD candidate Stephen Miner, who generously offered his time to help me gain a fundamental understanding of law to logic translations. I also want to thank Professor Scott Shapiro for his encouragement and my DUS, Professor Manohar, for his kindness and support throughout my time at Yale as an EECS major.

My experience as a Yale EECS student has been filled with wonderful professors and an incredible community of students by my side. I'm deeply grateful for the robust education I've received and for friends who've enriched every step. I'd also like to thank my family for their support and love throughout my education.

References

[BV03] Bart Verheij. Dialectical Argumentation with Argumentation Schemes: An Approach to Legal Logic. *Artificial Intelligence and Law*, 2003.

[CFA] Section 18 U.S. Code § 1030.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[dV23] Alex de Vries. The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194, 2023.

[OAD+23] Chukwufumnanya Ogbogu, Madeleine Abernot, Corentin Delacour, Aida Todri-Sanial, Sudeep Pasricha, and Partha Pratim Pande. Energy-efficient machine learning acceleration: From technologies to circuits and systems. In *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–8. IEEE, 2023.

[OK21] Orin Kirr. The Supreme Court Reigns in the CFAA in Van Buren. *Lawfare*, 2021.

[PLB16] Patricia L. Bellia. A Code-based Approach to Unauthorized Access Under the CFAA. *George Washington Law Review*, 1442, 2016.

[PMD09] Phan Ming Dung. Modular argumentation for modeling legal doctrines in common law of contract. *Artifical Intelligence and Law*, 2009.

[PMD10] Phan Ming Dung. A Logical Model of Private International Law. *Proceedings on the 10th International Conference on Deontic Logic in Computer Science*, pages 229–246, 2010.

[van21] Nathan Van Buren v. United States. 2021.

[VSC23] Roberto Verdecchia, June Sallou, and Luís Cruz. A systematic review of green ai. *WIREs Data Mining and Knowledge Discovery*, 13(4):e1507, 2023.