Logical Lease Litigation: Prolog and LLMs for Rental Law Compliance in New York

Sanskar Sehgal   Yanhong A. Liu

Stony Brook University, Stony Brook, NY 11794, USA

{sasehgal, liu}@cs.stonybrook.edu

Abstract

Legal cases require careful logical reasoning following the laws, whereas interactions with non-technical users must be in natural language. As an application combining logical reasoning using Prolog and natural language processing using large language models (LLMs), this paper presents a novel approach and system, LogicLease, to automate the analysis of landlord-tenant legal cases in the state of New York.

LogicLease determines compliance with relevant legal requirements by analyzing case descriptions and citing all relevant laws. It leverages LLMs for information extraction and Prolog for legal reasoning. By separating information extraction from legal reasoning, LogicLease achieves greater transparency and control over the legal logic applied to each case. We evaluate the accuracy, efficiency, and robustness of LogicLease through a series of tests, achieving 100% accuracy and an average processing time of 2.57 seconds. LogicLease presents advantages over state-of-the-art LLM-based legal analysis systems by providing clear, step-by-step reasoning, citing specific laws, and distinguishing itself by its ability to avoid hallucinations—a common issue in LLMs.

1 Introduction

Rental law compliance matters significantly to all rental residents. According to [2], more than 122.8 million households in the United States are renters. [5] estimates that there are over 1.1 million cases of landlords evicting tenants every year, representing an increase of 75% since 2021. Furthermore, 60% of eviction case defendants in 2023 were women, and despite making up less than one-third of renters, nearly half of eviction case defendants in 2023 were Black. [4] states that over seven million Americans are evicted from their homes every year, nearly 40% (2.7 to 3.2 million) of which are children. Additionally, in the United States, there is no guaranteed right to legal counsel in eviction proceedings. [16] estimates that as many as 90% of tenants facing eviction go to court unrepresented, putting them at a significant disadvantage. Consequently, up to 75% of tenants end up losing their eviction cases.

The legal domain's emphasis on meticulous analysis and transparent thought processes makes it an ideal candidate for utilizing logic-based systems over black-box approaches. Logic-based systems excel in providing clear and transparent reasoning, which is highly valued in legal contexts. While large language models (LLMs) are increasingly being considered for automating legal analysis, they are prone to hallucinations, which can lead to incorrect legal interpretations.

This paper presents LogicLease, a novel system specifically designed to automate the analysis of landlord-tenant legal cases in the state of New York, providing a transparent and reliable alternative to black-box methods.

Rental law compliance is crucial for the well-being of tenants, but navigating legal cases can be complex and time-consuming. With millions of eviction cases each year, many tenants face eviction without proper legal representation, resulting in a significant disadvantage.

LogicLease harnesses the strengths of Large Language Models (LLMs) for information extraction and Prolog for legal reasoning. It is designed to assess compliance with relevant legal requirements by analyzing case descriptions presented in natural language. LogicLease utilizes LLMs to parse the input, extracting essential details that can influence case outcomes. Subsequently, it employs a logic-based backend to generate clear, step-by-step reasoning and cite specific laws.

The accuracy, efficiency, and robustness of LogicLease were evaluated through a series of tests, demonstrating 100% accuracy and an average processing time of 2.57 seconds. By separating information extraction from legal reasoning, LogicLease ensures greater transparency and control over the legal logic applied to each case. Furthermore, LogicLease addresses the challenges of hallucinations and opacity common in LLMs, providing a reliable tool for landlord-tenant legal analysis in New York.

The rest of the paper is organized as follows. Section 2 describes the problem of reliable analysis of rental law compliance. In Section 3, we present the approach used in LogicLease for automating the analysis of landlord-tenant legal cases. Section 4 provides implementation details and evaluates the accuracy, efficiency, and robustness of LogicLease. Finally, Section 5 discusses related work and concludes the paper.

2   The need for legal compliance analysis

Millions of Americans face challenges in their homes, and those from marginalized communities are often disproportionately impacted by unfair housing practices. To address this gap, we developed LogicLease. LogicLease promises to provide free and highly accurate guidance on tenant-landlord issues. Our system is designed for simplicity, so anyone can navigate it, regardless of technical expertise. We prioritize accuracy to ensure users receive reliable information to confidently address their housing concerns.

When developing LogicLease, we established three primary requirements:

1. Natural Language Input: Users can describe their situation in plain English (or potentially another language). This could involve describing the issue (e.g., repairs not being made, rent increase concerns), relevant details (lease terms, dates), and any questions they have.

2. Rigorous Analysis for High Accuracy: The system uses LLMs to understand the user's situation and intent. It analyzes legal regulations, relevant case law, and best practices using the Prolog backend to provide accurate guidance. This might involve identifying the key legal issues involved, assessing the user's rights and responsibilities based on their location and lease agreement and considering potential solutions or next steps.

3. Natural Language Output: The system provides clear and actionable information tailored to the user's situation. This output is generated using a Prolog backend, where description of relevant laws is coded as strings associated with the rules. As the Prolog backend processes the rules and retrieve the specific law applicable to the case, it generates and prints the corresponding description to the output. This may include explanations of tenant rights or step-by-step guidance on how to address issues (e.g., contacting the landlord, or filing a complaint), presented in a way that is easy to understand, even for people with no legal background.

3 Combining logic programming and LLMs for legal analysis

LogicLease consists of four main components:

1. Driver Script: This component serves as the central coordinator, orchestrating the interaction between the other components. It takes a case description (lease agreement) as input and invokes the Natural Language Processing (NLP) and Prolog functionalities for analysis.

2. Natural Language Processing (NLP): LogicLease utilizes a large language model (LLM) out-of-the-box, without any additional training, to extract attribute-value pairs from the lease agreement. These pairs represent critical aspects of the case, such as whether the lease is signed or the duration of the rental period.

3. Prolog Knowledge Base: This component contains a set of Prolog rules that represent legal requirements for rentals, which we manually coded based on the New York Renters' Rights Handbook [12]. It contains an exhaustive set of rules about lease validity, rent stabilization, eviction, habitability, and more. LogicLease utilizes the attribute-value pairs extracted by the LLM as arguments for Prolog queries and uses the Prolog engine to evaluate compliance against these rules.

4. User Interface: The user interface of LogicLease provides users with an intuitive and user-friendly experience, enabling them to interact with the system seamlessly (Figure 1a). Users input the case description in natural language. Alternatively, they can also answer a series of dynamically generated questions related to the specific legal aspects of their case. These questions are designed to extract key attributes necessary for the legal analysis (Figure 1b). Users can provide answers using dropdown menus or text input fields. The user interface processes the information and invokes the legal analysis.
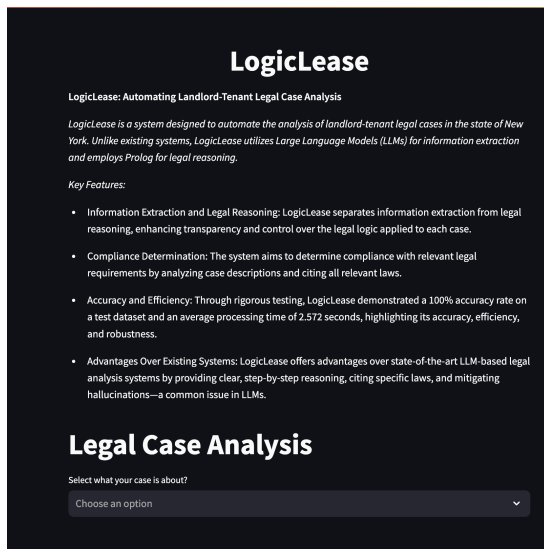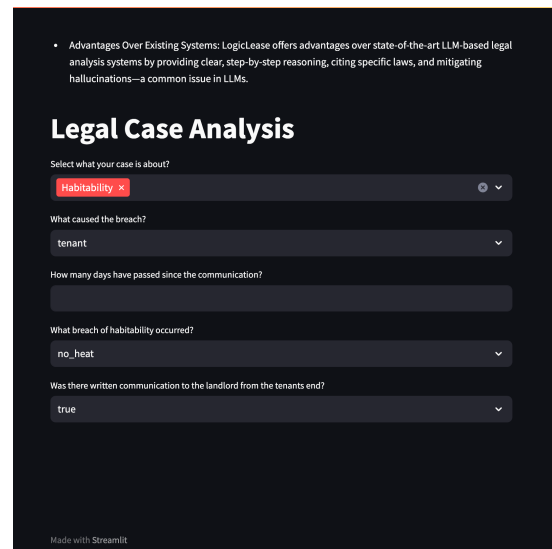
(a) User Interface of LogicLease   (b) Example of dynamically generated questions

Figure 1: LogicLease front-end interface

As shown in Figure 2, the design of LogicLease follows a structured process. Initially, the Driver Script receives a case description in natural language as input. Subsequently, it calls the API to the LLM, providing a system query that specifies the desired lease agreement aspects and the user query

containing the actual case description. The API call sends the queries to the LLM, retrieving a response containing extracted attribute-value pairs. The Driver Script parses this response and stores the extracted attribute-value pairs in a dictionary. Following this, the Driver Script invokes Prolog functions passing the dictionary containing the extracted attribute-value pairs. These functions translate the dictionary into a Prolog query based on the pre-defined Prolog rules in the knowledge base. The Prolog engine then evaluates the query, determining compliance with the lease agreement requirements. Finally, the results of the compliance check are displayed by the Driver Script.
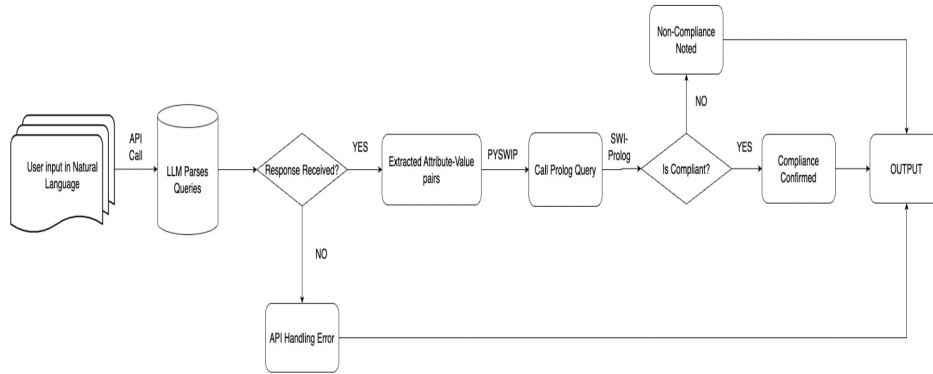

Figure 2: Design diagram of LogicLease

To illustrate the workflow of LogicLease and to facilitate a deeper understanding of the system, we provide an overview of the input processing, attribute-value pair extraction using an LLM, compliance checking with the Prolog backend system, and the final output.

User input of case in natural language. The input for a case is taken in natural language. An example is the following:

> In a rent-stabilized apartment in Albany, New York, David, a disabled tenant, faced eviction proceedings initiated by his landlord, Ms. Johnson, citing owner occupancy as the cause. Despite David's disability, he has been asked to vacate. The matter has not been presented before court, and hence does not have a court ruling yet.

This input is passed to the LLM for processing.

Attribute-value pairs extracted. The LLM parses the input text and outputs attribute-value pairs which are the core details of the case which influence the outcome. The extracted attribute-value pairs indicate that the eviction cause is "owner occupancy," there is no court ruling yet ("CourtRuling" is "false"), and the tenant is in a protected category ("TenantCategory" is "disabled"). These attribute-value pairs are stored in a dictionary in the following format.

EvictionCause : "owner_occupancy",

CourtRuling : "false",

Executioner : "null",

TenantCategory : "disabled",

Logic rules and queries in Prolog. As seen in Listing 1, the compliance with legal requirements in this specific case is determined by invoking the following Prolog predicate. Notably, the arguments precisely match the information extracted by the Large Language Model (LLM) from the case description.

```prolog
eviction(EvictionCause, CourtRuling, Executioner, TenantCategory) :-
    eviction_law,
    (CourtRuling == true ->
        (
            eviction_legal(EvictionCause),
            writeln('All conditions satisfied, eviction is lawful.'),
            nl
        )
    ;
    (
        eviction_warrant_execution(Executioner),
        \+ overrides(TenantCategory, EvictionCause),
        writeln('All conditions satisfied, eviction is lawful.'),
        nl
    )
).
```

Listing 1: Prolog rule for eviction compliance checking

The arguments of the predicate are instantiated based on the extracted attribute-value pairs stored in the dictionary. The call to the Prolog predicate is structured as follows:

```prolog
Prolog.query(eviction(owner_occupancy, false, null, disabled).)
```

The Prolog query evaluates the compliance of the landlord's actions with the relevant legal requirements.

Final output to user in natural language. The final output consists of a list of laws relevant to the case followed by a final judgment based on these laws. This transparent mechanism provides a clear rationale for each case. For our example, the output is:

1. Tenant with a lease is protected from eviction during the lease period if lease provisions and local laws are not violated.

2. Landlords must give formal notice before seeking legal possession of the apartment.

3. Eviction proceedings can be initiated for non-payment or significant lease violations.

4. Landlords of rent-regulated apartments may need DHCR approval for court proceedings.

5. Tenants should not ignore legal papers to avoid eviction.

6. Legal eviction requires court proceeding and judgment of possession.

7. Landlords cannot evict tenants unlawfully or by force.

8. Tenant evicted unlawfully can recover triple damages.

9. Additional rules protect certain groups from eviction.

Tenant is in protected category, eviction for owner occupancy unlawful.

The output confirms that the eviction for owner occupancy is unlawful given the tenant's protected category status and the absence of a court ruling.

4   Implementation and evaluation

We have developed a complete implementation of the LogicLease system, covering the full New York State landlord-tenant legal framework. We successfully applied our system to a series of 10 test cases, representing various scenarios encountered in landlord-tenant disputes. To evaluate the system's performance, we employ a multi-pronged approach, assessing its accuracy, efficiency, and robustness across various dimensions.

4.1   Implementation

LogicLease is implemented using a combination of Python and SWI-Prolog [21, 17]. Python is used to facilitate interactions among different components, utilizing libraries such as PYSWIP to establish communication between the Python script and the Prolog engine (SWIPL), while Streamlit is employed to create a user-friendly front-end interface. Additionally, llamaapi [10] is used to manage interactions with the Large Language Model (LLM) model LLaMA [18], while Python libraries like json and ast are employed for processing the output received from LLaMA.

The Prolog backend serves as the foundation for legal reasoning, with custom clauses defining the relevant laws and procedures. The implementation of the system followed a structured approach, ensuring modularity and ease of maintenance.

Notably, LogicLease incorporated defeasible logic [19, 20, 11] within the Prolog component, enabling the system to handle situations where one legal principle takes precedence over another under specific circumstances.

By separating information extraction (NLP) from legal reasoning (Prolog), LogicLease achieved greater transparency and control over the legal reasoning applied to each case. This approach is particularly important for legal professionals who require a clear understanding of the system's reasoning process, while ensuring there are no hallucinations or snowballing effects.

The total size is approximately 500 lines of Prolog code and an additional 400 lines of Python code.

All experiments and measurements were conducted on a macOS system featuring an Apple Silicon M2 processor, 8GB of RAM, and a 256GB SSD. The system was running macOS Monterey version 12.5, with Python 3.9.12, Prolog SWIPL 9.2.2, PYSWIP 0.2.11, llamaapi version 0.1.36, and Streamlit version 1.24.1.

4.2   Accuracy

To evaluate LogicLease's effectiveness, we manually compiled a dataset of lease litigation cases in New York. This dataset includes a mix of real-world cases (condensed for efficiency) and fictional scenarios we created. Due to limited API credits available for using the LLM, the system restricted the number of API calls made during the processing of legal cases. This limitation necessitated significant shortening of the text within the legal documents to fit within the allowed API usage. Despite these constraints, the resulting dataset of 10 cases effectively demonstrates the system's accuracy and usability in analyzing and responding to legal scenarios.

Encouragingly, in all ten cases tested, the LLM functioned effectively. It accurately extracted relevant details from each case description and successfully transferred this information to the Prolog backend system. The Prolog system, in turn, flawlessly interpreted the queries and delivered final verdicts on the legal issues presented. Although the dataset is currently small, the system's 100% accuracy on this dataset helps build trust in the system.

To evaluate the accuracy of the system's reasoning, we conducted human-in-the-loop evaluation. This involved manually reviewing the system's output for each case. Legal resources such as handbooks and online legal forums (e.g., on Reddit) were used to verify the system's determinations. This process helped identify potential biases in the LLM's interpretation of the case details or errors within the Prolog code's reasoning logic.

4.3 Efficiency

We measured the average processing time per case, including the LLM extraction and the logic-based compliance check. The processing times are shown in Table 1. The main bottleneck is the API call to the LLM Llama, which is not directly under our control. However, solutions such as caching frequently accessed legal information can be explored.

While the LLM API call (to Llama) currently represents the performance bottleneck compared to Prolog's minimal processing time, the good news is that the average case processing time remains user-friendly at only 2.572 seconds.

It also helps if the case description passed to Llama is as concise as possible. Not surprisingly, the time taken for Llama to respond is directly proportional to the length of the query string passed. In fact, the system exhibits a strong positive correlation (approx 68%) between the length of the query string and Llama's response time.

Test Case	Total Time	Prolog Running Time
1	2.693	0.00012
2	2.462	0.00018
3	2.412	0.00021
4	3.156	0.00120
5	3.324	0.00130
6	2.270	0.00090
7	2.155	0.00050
8	2.215	0.00042
9	2.551	0.00031
10	2.486	0.00010
Average	**2.572**	**0.00011**

Table 1: Running times, in seconds, for all test cases

4.4 Robustness

To highlight the superiority and robustness of our system compared to existing solutions, we present a comparison with state-of-the-art Large Language Models (LLMs) such as ChatGPT [13] and Gemini [3]. These LLMs appear to favor the landlord in most cases, as shown by their outputs for the running example shown in Figures 4 and 5. Neither of these LLMs reached the correct conclusion for this example
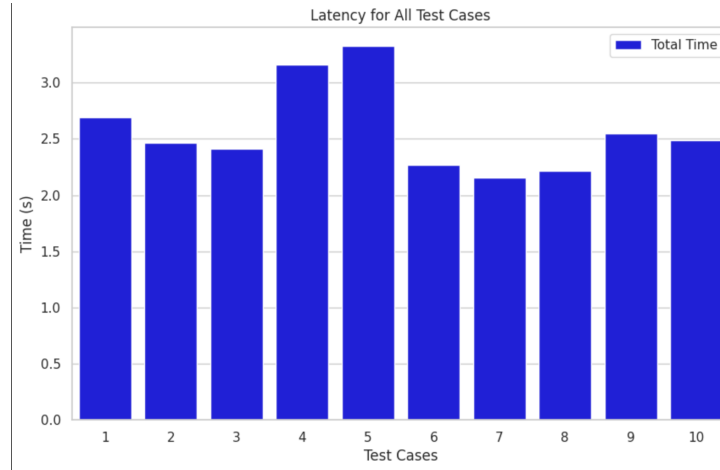
Figure 3: Total Time Taken for Each Test Case

case. According to New York law, the situation clearly favors the tenant if they live in a rent-stabilized apartment and belong to a protected category. For our test dataset, these LLMs did not reach the correct conclusion in three out of ten cases. This underscores the importance of our system and highlights the unreliability of LLMs for this application.
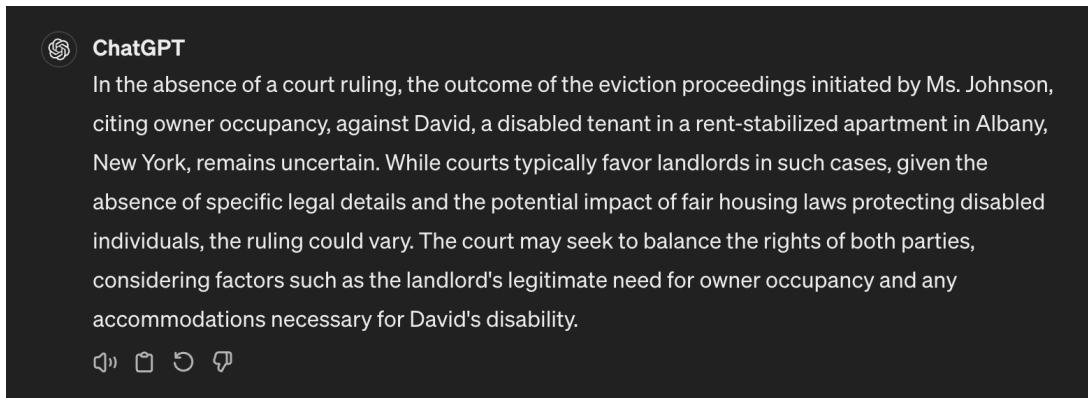

Figure 4: ChatGPT output

5   Related work and conclusion

Existing systems for legal analysis employ various techniques, including text classification, machine learning, and rule-based reasoning [1, 9, 6, 7, 8]. However, these approaches often have limitations in handling nuanced legal reasoning.

Text classification systems such as Legal-Document-Classifier [1] and LexNLP [9] can categorize legal documents based on keywords and named entities but lack the ability to perform comprehensive legal reasoning.

Machine learning-based systems like Kira [6] and LawGeex [7] can extract key terms and identify potential issues in contracts, and models like Lex Machina [8] can predict legal outcomes with some
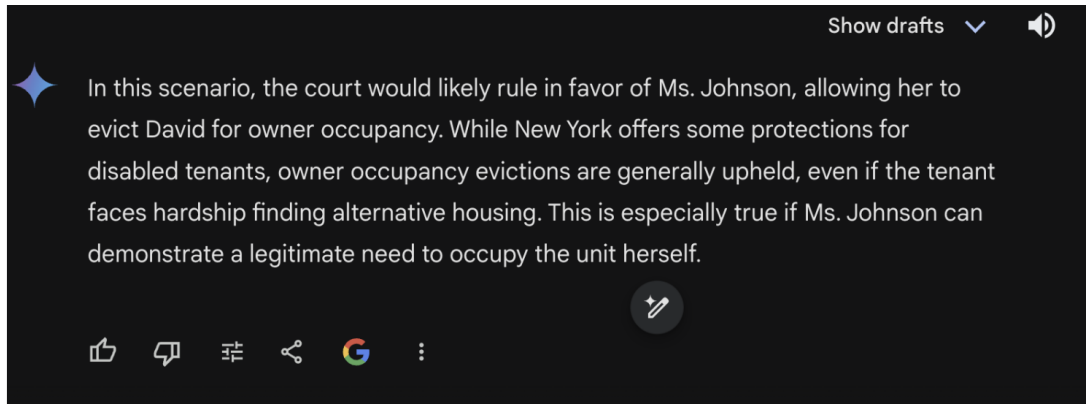
Figure 5: Gemini output

accuracy. However, these systems often operate as black boxes, raising concerns about transparency and fairness. They may also be limited by data quality and biases.

Rule-based legal reasoning systems like PROLEG [14, 15] offer support for judges in civil litigation by incorporating predefined rules and handling uncertainty. However, their complexity can pose challenges for users.

In contrast, we successfully developed a system for analyzing landlord-tenant disputes in New York State by leveraging Large Language Models (LLMs) for information extraction and Prolog for legal reasoning. Achieving high accuracy and efficiency, the system offers several advantages over existing LLM-based legal analysis systems.

By separating information extraction from legal reasoning, the system achieves greater transparency and control over the legal logic applied to each case. Additionally, the use of Prolog enables the implementation of defeasible logic [19, 20], allowing the system to handle nuanced legal reasoning, such as resolving conflicting legal principles and dealing with uncertain or incomplete information [11].

In conclusion, this work demonstrates the potential of combining Large Language Models (LLMs) and logic-based reasoning to create innovative tools for legal analysis. By addressing the limitations of existing approaches, LogicLease paves the way for more sophisticated and transparent systems in the field of legal technology.

Future work includes expanding the system's capabilities by employing techniques for caching frequently accessed legal information. Additionally, improving the testing process and adding more test cases will ensure better coverage and reliability of the system. Open-sourcing the code could encourage further development and broader adoption within the legal domain.

References

[1] Andrei-Marius Avram, Vasile Pais & Dan Tufis (2021): *PyEuroVoc: A Tool for Multilingual Legal Document Classification with EuroVoc Descriptors.* ArXiv preprint arXiv:2108.01139.

[2] Drew Desilver (2021): *As national eviction ban expires, a look at who rents and who owns in the US.* `https://www.pewresearch.org/short-reads/2021/08/02/as-national-eviction-ban-expires-a-look-at-who-rents-and-who-owns-in-the-u-s/`.

[3] Google (2024): *Gemini Large Language Model.* Large language model. Available at `https://gemini.google.com/app`.

[4] Nick Graetz, Carl Gershenson, Peter Hepburn, Sonya R Porter, Danielle H Sandler & Matthew Desmond (2023): *A comprehensive demographic profile of the US evicted population*. Proceedings of the National Academy of Sciences 120(41), doi:10.1073/pnas.2305860120.

[5] Peter Hepburn, Olivia Jin, Joe Fish, Emily Lemmerman, Anne Kat Alexander & Matthew Desmond (2022): *Preliminary analysis: Eviction filing patterns in 2021*. `https://evictionlab.org/us-eviction-filing-patterns-2021/`.

[6] KiraSystems (2015): *Kira*. `https://kirasystems.com/`.

[7] LawGeex (2014): *LawGeex*. `https://www.lawgeex.com`.

[8] Lemley Mark, Gregory George & Walker Joshua (2006): *Lex Machina*. `https://lexmachina.com/`.

[9] LexPredict (2024): *LexNLP by LexPredict*. `https://github.com/LexPredict/lexpredict-lexnlp`.

[10] llamaapi, Reis, Eduardo & Hu, Jesse (2023): *llamaapi-python*. `https://github.com/llamaapi/llamaapi-python`.

[11] Jason Morris (2024): *Rules as Code Demonstration in SWISH and s(CASP)*. `https://swish.swi-prolog.org/p/r34v2.swinb`. Accessed Mar. 17, 2024.

[12] New York State Attorney General (2024): *Tenants' Rights Guide*. `https://ag.ny.gov/sites/default/files/tenants_rights.pdf`.

[13] OpenAI (2023): *ChatGPT (Mar 14 version)*. Large language model. Available at `https://chat.openai.com/chat`.

[14] Ken Satoh, Kento Asai, Takamune Kogawa, Masahiro Kubota, Megumi Nakamura, Yoshiaki Nishigai, Kei Shirakawa & Chiaki Takano (2010): *PROLEG: an implementation of the presupposed ultimate fact theory of Japanese civil code by PROLOG technology*. In: *JSAI international symposium on artificial intelligence*, Springer, pp. 153–164, doi:10.1007/978-3-642-25655-4_14.

[15] Ken Satoh, Kento Asai, Takamune Kogawa, Masahiro Kubota, Megumi Nakamura, Yoshiaki Nishigai, Kei Shirakawa & Chiaki Takano (2011): *PROLEG: An Implementation of the Presupposed Ultimate Fact Theory of Japanese Civil Code by PROLOG Technology*. In Takashi Onada, Daisuke Bekki & Elin McCready, editors: *New Frontiers in Artificial Intelligence*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 153–164, doi:10.1007/978-3-642-25655-4_14.

[16] Heidi Schultheis & Caitlin Rooney (2019): *A right to counsel is a right to a fighting chance*. `https://www.americanprogress.org/article/right-counsel-right-fighting-chance/`.

[17] (2024): *SWI Prolog*. `https://www.swi-prolog.org`. Accessed Mar. 17, 2024.

[18] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar et al. (2023): *LlaMa: Open and efficient foundation language models*, doi:10.48550/arXiv.2302.13971. ArXiv preprint arXiv:2302.13971.

[19] Hui Wan, Benjamin Grosof, Michael Kifer, Paul Fodor & Senlin Liang (2009): *Logic programming with defaults and argumentation theories*. In: *Logic Programming: 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings 25*, Springer, pp. 432–448, doi:10.1007/978-3-642-02846-5_35.

[20] Hui Wan, Michael Kifer & Benjamin Grosof (2015): *Defeasibility in answer set programs with defaults and argumentation rules*. Semantic Web 6(1), pp. 81–98, doi:10.3233/SW-140140.

[21] Jan Wielemaker, Tom Schrijvers, Markus Triska & Torbjörn Lager (2012): *SWI-Prolog*. Theory and Practice of Logic Programming 12(1-2), pp. 67–96, doi:10.1017/S1471068411000494.